DESIGN AUTOMATION ALGORITHMS FOR ADVANCED LITHOGRAPHY

BY

ZIGANG XIAO

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Doctoral Committee:

Professor Martin D.F. Wong, Chair
Professor Rob A. Rutenbar
Professor Deming Chen
Professor Xiuling Li

ABSTRACT

In circuit manufacturing, as the technology nodes keep shrinking, conventional 193 nm immersion lithography (193i) has reached its printability limit. To continue the scaling with Moore's law, different kinds of advanced lithography have been proposed, such as multiple patterning lithography (MPL), extreme ultraviolet (EUV), electron beam lithography (EBL) and directed self-assembly (DSA). While these new technologies create enormous opportunities, they also pose great design challenges due to their unique process characteristics and stringent constraints. In order to smoothly adopt these advanced lithography technologies in integrated circuit (IC) fabrication, effective electronic design automation (EDA) algorithms must be designed and integrated into computer-aided design (CAD) tools to address the underlying design constraints and help the circuit designer to better facilitate the lithography process. In this thesis, we focus on algorithmic design and efficient implementation of EDA algorithm for advanced lithography, including directed self-assembly (DSA) and self-aligned double patterning (SADP), to conquer the physical challenges and improve the manufacturing yield.

The first advanced lithography technology we explore is self-aligned double patterning (SADP). SADP has the significant advantage over traditional litho-etch-litho-etch (LELE) double patterning in its ability to eliminate overlay, making it a preferable DPL choice for the 14 nm technology node. As in any DPL technology, layout decomposition is the key problem. While the layout decomposition problem for LELE DPL has been well studied in the literature, only a few attempts have been made for the SADP layout decomposition problem. This thesis studies the SADP decomposition problem in different scenarios.

SADP has been successfully deployed in 1D patterns and has several applications; however, applying it to 2D patterns turns out to be much more difficult. All previous exact algorithms were based on computationally expen-

sive methods such as SAT or ILP. Other previous algorithms were heuristics without a guarantee that an overlay-free solution can be found even if one exists. The SADP decomposition problem on general 2D layout is proven to be NP-complete. However, we show that if we restrict the overlay, the problem is polynomial-time solvable, and present an exact algorithm to determine if a given 2D layout has a no-overlay SADP decomposition.

When designing the layout decomposition algorithms, it is usually useful to take the layout structure into consideration. As most of the current IC layouts adopt a row-based standard cell design style, we can take advantage of its characteristics and design more efficient algorithms compared to the algorithms for general 2D patterns. In particular, the fixed widths of standard cells and power tracks on top and bottom of cells suggest that improvements can be made over the algorithms for general decomposition problem. We present a shortest-path based polynomial time SADP decomposition algorithm for row-based standard cell layout that efficiently finds decompositions with minimum overlay violations. Our proposed algorithm takes advantage of the fixed width of the cells and the alternating power tracks between the rows to limit the possible decompositions and thus achieve high efficiency.

The next advanced lithography technology we discuss in the thesis is directed self-assembly (DSA). Block copolymer directed self-assembly (DSA) is a promising technique for patterning contact holes and vias in 7 nm technology nodes. To pattern contacts/vias with DSA, guiding templates are usually printed first with conventional lithography (193i) that has a coarser pitch resolution. Contact holes are then patterned with DSA process. The guiding templates play the role of defining the DSA patterns, which have a finer resolution than the templates. As a result, different patterns can be obtained through controlling the templates. It is shown that DSA lithography is very promising in patterning contacts/vias in 7 nm technology node. However, to utilize DSA for full-chip manufacturing, EDA for DSA must be fully explored because EDA is the key enabler for manufacturing, and the EDA research for DSA is still lagging behind.

To pattern the contact layer with DSA, we must ensure that all the contacts in the layout require only feasible DSA templates. Nevertheless, the original layout may not be designed in a DSA-friendly way. However, even with an optimized library, infeasible templates may be introduced after the physical design phase. We propose a simulated-annealing (SA) based scheme

to perform full-chip level contact layer optimization. According to the experimental results, the DSA conflicts in the contact layer are reduced by close to 90% on average after applying the proposed optimization algorithm.

It is a current trend that industry is transiting from the random 2D designs to highly regular 1D gridded designs for sub-20 nm nodes and fabricating circuit designs with print-cut technology. In this process, the randomly distributed cuts may be too dense to be printed by single patterning lithography. DSA has proven its success in contact hole patterning, and can be easily expanded to cut printing for 1D gridded designs. Nevertheless, the irregular distribution of cuts still presents a great challenge for DSA, as the self-assembly process usually forms regular patterns. As a result, the cut layer must be optimized for the DSA process. To address the above problem, we propose an efficient algorithm to optimize cut layers without hurting the original circuit logic. Our work utilizes a technique called 'line-end extension' to move the cuts and extend the functional wires without changing the original functionality of the circuit. Consequently, the cuts can be redistributed and grouped into valid DSA templates.

Multiple patterning lithography has been widely adopted for today's circuit manufacturing. However, increasing the number of masks will make the manufacturing process more expensive. By incorporating DSA into the multiple patterning process, it is possible to reduce the number of masks and achieve a cost-effective solution. We study the decomposition problem for the contact layer in row-based standard cell layout with DSA-MP complementary lithography. We explore several heuristic-based approaches, and propose an algorithm that decomposes a standard cell row optimally in polynomial-time. Our experiments show that our algorithm is guaranteed to find a minimum cost solution if one exists, while the heuristic cannot or only finds a sub-optimal solution. Our results show that the DSA-MP complementary approach is very promising for the future advanced nodes.

As in any lithography technique, the process variation control and proximity correction are the most important issues. As the DSA templates are patterned by conventional lithography, the patterned templates are prone to deviate from mask shapes due to process variations, which will ultimately affect the contacts after the DSA process even for the same type of template. Therefore, in order to enable the DSA technology in contact/via layer printing, it is extremely important to accurately model and detect hotspots, as

well as estimate the contact pitch and locations during the verification phase. We propose a machine learning based design automation framework for DSA verification. A novel DSA model and a set of features are included. We implemented the proposed ML-based flow and performed extensive experiments on comparing the performances of learning algorithms and features. The experimental results show that our approach is much more efficient than the traditional approach, and can produce highly accurate results.

To my father, Xianzhi Xiao,
and my mother, Liping Yuan,
for their endless love and support.

ACKNOWLEDGMENTS

I would like to devote my deepest gratitude towards my advisor, Prof. Martin D.F. Wong. Without this support, I would not have been able to proceed and finish my PhD work. I have always been impressed and learned a lot from his thoughtful advice and wisdom.

I am wholeheartedly thankful to my doctoral committee, Prof. Deming Chen, Prof. Rob Rutenbar and Prof. Xiuling Li. Their comments and insightful suggestions have proven to be extremely valuable in my thesis.

I am especially honored to have the chance to work with Prof. Rutenbar as one of his teaching assistants. His deep knowledge of the subject, compassionate teaching, and empathy for students have always inspired me to strive for excellence. I am also thankful to my fellow teaching assistants, Mr. Chen-Hsuan 'Adonis' Chen and Mr. Nicholas Chen, for their support and professional service during the course.

I could not be more grateful to everyone who has worked with me during my PhD career for their support. I would like to give special thanks to Dr. Hongbo Zhang, who has provided invaluable help in both of my academic and personal life. I would also like to thank Dr. Yuelin Du, Prof. H.-S. Philip Wong and Dr. Linda He Yi for providing me enormous advice and collaboration for my research in DSA. I owe my deepest gratitude to Ms. Leslie Hwang, who always provided me great support whenever I encountered difficulties. I would also like to thank Mr. Haitong Tian, Dr. Ting Yu, Dr. Pei-Ci Wu, Mr. Daifeng Guo, Mr. Chun-Xun Lin, Dr. Qiang Ma, Dr. Tan Yan, Dr. Lijuan Luo and Mr. Tsung-Wei Huang. Without them, my PhD life cannot have been so colorful and enjoyable.

Finally, I owe my deepest gratitude to my parents, who have always been supportive of my decisions, including pursuing a PhD in the US. Their love and understanding keep me moving forward, and words cannot express my gratitude.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

193i	193 nm ArF Immersion Lithography
1D	1 Dimensional
2D	2 Dimensional
2SAT	2-Satisfiability
2CNF	2-Conjunctive Normal Form
AAM	Aux-Aux-Merge
ANN	Artificial Neural Network
BFS	Breadth-First Search
Bagging	Bootstrap Aggregating
CAD	Computer-Aided Design
CMOS	Complementary Metal-Oxide-Semiconductor
CAM	Core-Aux-Merge
CAR	Core-Aux-Removal
CCM	Core-Core-Merge
CF	Color-First
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DFM	Design For Manufacturing
DP	Double Patterning
DPL	Double Patterning Lithography

DSA	Directed Self-Assembly
DTW	Dynamic Time-Warping
DTCO	Design Technology Co-Optimization
DUV	Deep Ultraviolet
EBL	Electron Beam Lithography
EDA	Electronic Design Automation
EPE	Edge Placement Error
EUV	Extreme Ultraviolet
FPT	Fixed-Parameter Tractable
GB	Gigabytes
GF	Group-First
GHz	Gigahertz
HOG	Histogram or Oriented Gradients
IC	Integrated Circuit
ILP	Integer Linear Programming
LELE	Litho-Etch-Litho-Etch
LI	Local Interconnect
LWR	Line Width Roughness
M1	Metal 1
M2	Metal 2
ML	Machine Learning
MP	Multiple Patterning
MPL	Multiple Patterning Lithography
MRC	Manufacturing Rule Checks
NGL	Next Generation Lithography
NMOS	N-type Metal-Oxide-Semiconductor
NP	Non-deterministic Polynomial-time

OPC	Optical Proximity Correction
OPT	Optimal
PMOS	P-type Metal-Oxide-Semiconductor
RAM	Random Access Memory
RBF	Radial Basis Function
RET	Resolution Enhancement Technique
RF	Random Forest
RMSE	Root Mean Square Error
SA	Simulated Annealing
SADP	Self-Aligned Double Patterning
SAT	Satisfiability
SCFT	Self-Consistent Field Theory
SEM	Scanning Electron Microscope
SID	Spacer-Is-Dieletric
SIFT	Scale-Invariant Feature Transform
SVM	Support Vector Machine
SVR	Support Vector Regression
TPL	Triple Patterning Lithography
VLSI	Very-Large-Scale Integration

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

In the semiconductor industry, the design and manufacturing communities had been enjoying being able to optimize on their area-specific goals without excessive dependency on each other. In particular, designers had been able to rely on electronic design automation (EDA) tools to perform computer-aided design (CAD), such as using automatic place and route system to minimize the design area. The designers expected the process scaling would keep up with the feature size shrinking and thus bring substantial reduction in chip size and power. Indeed, for a long time manufacturers had successfully maintained delivering new process technology to scale with the design needs. As a result, designers had been able to achieve area and performance improvement without much of the material and process involved in the design flow.

Nevertheless, scaling CMOS technology at the speed dictated by Moore's law has become increasingly difficult. The reason is that the lithography process has lagged far behind the technology node. While the technology node has shrunk to sub-20 nm in recent years, the underlying process technology, 193 nm immersion lithography (193i), has not been improved. Even though resolution enhancement techniques (RET) such as optical proximity correction (OPC) have been proposed to enable printing small features with the coarse resolution of 193i, the printability limit of 193i is finally reached at 28 nm node.

To keep up with Moore's law, a number of advanced lithography technologies have been proposed in recent years, such as multiple patterning lithography (MPL), including self-aligned double patterning (SADP), triple patterning (TPL) and beyond, directed self-assembly (DSA) and extreme ultraviolet (EUV). While these new technologies create enormous opportu-

nities, they also pose great design challenges due to their unique process characteristics and stringent constraints. Instead of only consulting EDA tools for design for manufacturing (DFM) feedback, the designers have to consider the process constraints at the early stage of the design. Eventually, design-technology co-optimization (DTCO) becomes the key enabling factor for CMOS scaling. As a result, the EDA engines have to understand the underlying process technology, and corresponding design automation algorithms need to be designed to address the special issues and process rules. This motivates us to explore and study the topic of design automation algorithms for advanced lithography, and it is the main theme of the dissertation.

1.2 Overview of This Dissertation

In this dissertation, we present our research results on two of the most promising advanced lithography technologies: self-aligned double patterning (SADP) and directed self-assembly (DSA).

SADP has a significant advantage over conventional LELE (Litho-Etch-Litho-Etch) DPL in its ability to eliminate overlay, making it a preferable DPL choice for the 14 nm technology node. As in any DPL technology, layout decomposition is the key problem. While the layout decomposition problem for LELE DPL has been well-studied in the literature, only a few attempts have been made to solve the SADP layout decomposition problem. SADP has been successfully deployed in 1D patterns and has several applications. However, applying it to 2D patterns turns out to be much more difficult. The previous exact algorithms were based on computationally expensive methods such as SAT or ILP [1], [2]. Other existing works were heuristics without a guarantee that an overlay-free solution can be found even if one exists [3]–[5]. In Chapter 2, we present a polynomial-time exact (optimal) algorithm to determine if a given layout has SADP decompositions that do not have overlay at critical edges. Compared to the previous work, our proposed algorithm can solve all the test cases in a reasonable time.

In Chapter 3, we address the SADP decomposition on row-based standard cell layout and propose a polynomial-time algorithm. Our objective is to minimize the overall overlay violations. Although SADP decomposition has been shown to be NP-hard in general, we show that it can be solved in poly-

nomial time when the layout is row-based standard cells. The experimental results with industrial level standard cells demonstrate our method can solve large scale problems in a relatively short time. As row-based standard cell design is a major choice in ASIC design, our approach is expected to find its potential in the manufacturing industry for the sub-20nm technology node.

It is shown that DSA lithography is very promising in patterning contacts/vias in 7 nm technology node [6]–[8]. However, to utilize DSA for full-chip manufacturing, EDA for DSA must be fully explored as EDA is the key enabler for manufacturing, and the EDA research for DSA is still lagging behind. In Chapter 4, we discuss contact layer and cut layer optimization problems for DSA. To pattern the contact layer with DSA, we must ensure that all the contacts in the layout require only feasible DSA templates. As a result, the original layout must be designed in a DSA-friendly way. A previous work on contact layer optimization focused on cell level optimization and aimed to minimize the total templates used in the standard cell library [9]. However, even with an optimized library, infeasible templates may be introduced after the physical design phase. We propose a simulated-annealing (SA) based scheme to perform full-chip level contact layer optimization. According to the experimental results, the DSA conflicts in contact layer are reduced by close to 90% on average after applying the proposed optimization algorithm.

Recently, the industry has been transiting from the random 2D designs to highly regular 1D gridded designs for sub-20nm nodes and fabricating circuits designs with print-cut technology. In this design style, randomly distributed cuts are the major challenge as they may be too dense to be printed by single patterning lithography. DSA has proven its success in contact hole patterning, and can be easily expanded to cut printing for 1D gridded designs. Nevertheless, the irregular distribution of cuts still presents great challenge for DSA, as the self-assembly process usually forms regular patterns. As a result, the cut layer must be optimized for the DSA process. To address the above problem, we propose an efficient algorithm to optimize cut layers without hurting the original circuit logic in Chapter 4. Our work utilizes a technique called 'line-end extension' to move the cuts and extend the functional metal wires without changing the original functionality of the circuit. Consequently, the cuts can be redistributed and grouped into valid DSA templates. Our experimental result showed that the final cut conflicts

are resolved with relatively small costs after the optimization.

Multiple patterning lithography (MPL) has been widely adopted for today's circuit manufacturing. However, increasing the number of masks will make the manufacturing process more expensive. More importantly, towards 7 nm technology node, the accumulated overlay in multiple patterning (MP) will cause unacceptable edge placement error (EPE). Directed self-assembly (DSA) has been shown to be an effective lithography technology that can pattern contact/via/cuts with high throughput and low cost. DSA is currently aiming at 7 nm technology, where the guiding template generation needs either double patterning EUV (extreme ultraviolet) or multiple patterning DUV (deep ultraviolet) process. By incorporating DSA into the multiple patterning process, it is possible to reduce the number of masks and achieve a cost-effective solution. In Chapter 5, we study the contact layer decomposition problem in row-based standard cell layout with DSA-MP complementary lithography. We explore several heuristic-based approaches, and propose an algorithm that decomposes a standard cell row optimally in polynomial-time. Our experiments show that our algorithm is guaranteed to find a minimum cost solution if one exists, while the heuristics can only find sub-optimal solutions. Our results show that the DSA-MP complementary approach is very promising for the future advanced nodes.

In DSA lithography, as the templates are patterned by conventional lithography (193i), their shapes may vary due to the process variations, which will ultimately affect the contacts/vias even for the same type of template. Although a rigorous DSA simulation can link the guiding template generation process to the DSA pattern formation [10]–[13], the extremely low efficiency makes it barely suitable for adoption in the full-chip level implementation, *e.g.*, DSA-aware OPC and lithography verification. Therefore, effective DSA verification methods are urgently needed. Meanwhile, the machine learning based technique has been shown to be effective in lithography verification tasks, such as hotspot detection. In Chapter 6, we propose a machine learning based design automation framework for DSA verification. A novel DSA model and a set of features are included. Following the proposed framework, we formulate the DSA hotspot detection and contact pitch and location prediction problems and address them using machine learning techniques. Extensive experiments are performed to compare the performances of learning algorithms and features. For hotspot detection, we can achieve 93% accu-

racy with the benchmarks. For contact pitch and location prediction, we can achieve an average RMSE (root mean square error) of below 0.5. The experimental results also showed that our approach is much more efficient than the traditional approach.

CHAPTER 2

SADP DECOMPOSITION FOR 2D LAYOUT

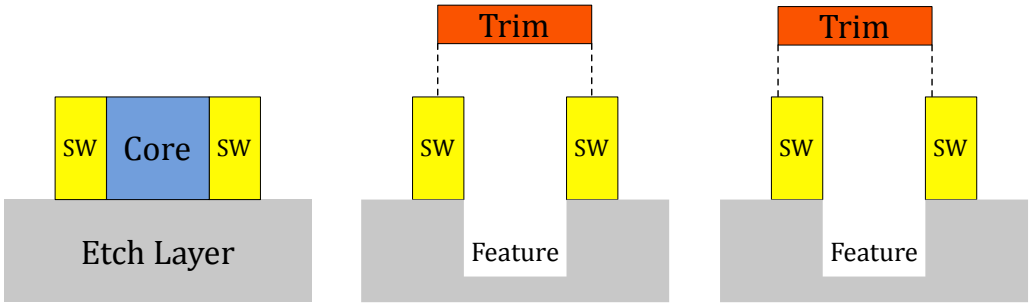
2.1 Introduction

As the feature size keeps shrinking and new lithography technologies such as extreme ultraviolet (EUV) have been further delayed, double patterning lithography (DPL) technologies have become the most promising process for today's sub-32 nm technology nodes with the 193 nm micro-lithography [14]. The conventional DPL such as LELE (litho-etch-litho-etch) that splits the feature patterns into two separate exposures, has been receiving much attention and there are quite a few works on it [15]–[19]. However, a fundamental problem that conventional DPL technologies suffer from is the overlay issue. Due to problems such as potential mask misplacement and wafer thickness variations, the features printed may have overlay. This results in hazardous consequence such as connection failures or short circuit and ultimately leads to serious yield degradation.

Recently, a new double patterning technology called self-aligned double patterning (SADP) has drawn lots of attention. Compared to LELE, SADP has many advantages such as better overlay tolerance and lower line width roughness (LWR). It is anticipated that SADP will be utilized in more advanced technology nodes with its smaller spacing rules and more flexible process control due its intrinsic alignment property [20]. In SADP, the overlay can be avoided by carefully *decomposing the layout*, *i.e.*, splitting the features to be patterned into two masks such that the target layout can be printed exactly. The first mask of SADP is called *core* mask, which is used to define locations of *sidewall spacers*. The second mask may be a *trim* mask that is used to 'print out' the feature patterns, or a *cut* (or block) mask that is used to 'cut out' the features, depending on the underlying technology (we adopt the trim mask style in this chapter). The sidewall patterns generated

after core mask are used to mask out the regions they are covering from the trim mask; *i.e.*, only the regions that are covered by the trim mask but not covered by sidewalls will be etched out after the trim mask phase. Thus, the sidewalls are essentially *defining the features*.

Figure 2.1 shows the crosscut view of printing a single rectangular feature using SADP process.[1] After core mask, a core pattern is generated and then sidewall patterns are deposited along the boundaries of the core pattern (Figure 2.1(a)). Afterwards, the core pattern is removed. A trim pattern will then be used to etch out the target feature (Figure 2.1(b)). Note that only the area that is *covered by trim mask but not covered by sidewalls* is patterned. Figure 2.1(c) illustrates how SADP can help to avoid overlay: even if the trim mask is misaligned with the core mask (within a certain extent), the target feature can still be printed exactly at the desired position as long as the amount of misalignment is bounded within the sidewall.



(a) Core patterns and sidewalls deposited.

(b) Trim pattern and feature produced.

(c) Trim pattern misaligned.

Figure 2.1: Crosscut view of printing a single feature using SADP process. 'SW' denotes sidewall.

It is true that SADP decomposition is not as intuitive and straightforward as LELE is. However, as [20] pointed out, SADP has many advantages over LELE such as higher scaling capability. Thus, more effort should be spent on the design automation algorithms to relieve the designers from the unique design challenges presented by SADP.

[1]There are two types of SADP process: positive tone and negative tone. Since the positive tone process is more flexible, it is thus considered as a more promising technique. We assume positive tone process is used in our work. Please refer to [21] for more detail.

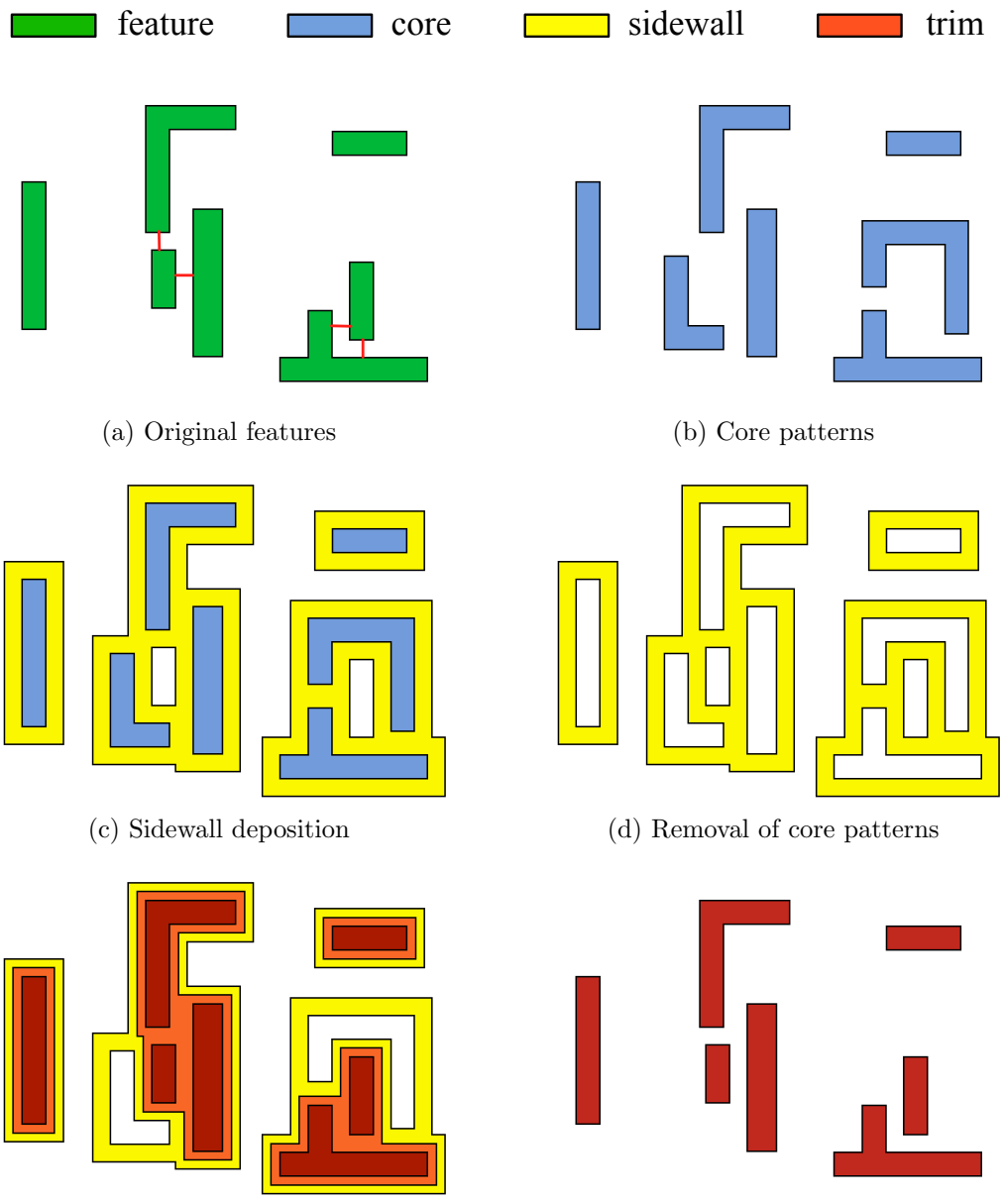
2.1.1 Applying SADP to Multiple Features

Although SADP has been successfully deployed in 1D patterns and has several applications, applying it to 2D patterns is relatively new and needs further attention. Unlike LELE DPL, the core patterns and trim patterns in SADP are not mapped to the original features directly. For a single feature, it is easy to use core mask to generate a core pattern that has the same shape as the feature, deposit the sidewall and use trim pattern to etch out the desired feature as in the previous example. It may not be as obvious when there are multiple features.

Figure 2.2 shows an example of printing multiple features using SADP process. In the target layout (Figure 2.2(a)), features are denoted in green shapes. Note that some of the features are too close according to the process rules and cannot be printed simultaneously using only a core mask. The conflicting pairs are marked with red lines. With SADP, they can be printed and overlay can be avoided. Figure 2.2(b) shows the core patterns denoted by the blue rectilinear shapes after the core mask phase. Notice that for some of the features, there may not be corresponding core patterns. Instead, some extra core patterns that do not correspond to any features are used to generate sidewalls to define the remaining features. The extra cores play a vital part in SADP layout decomposition. We refer to this type of core pattern as an *auxiliary core*, as it is essentially helping to generate sidewalls to avoid overlay.

We proceed by depositing sidewall patterns around the boundaries of the core patterns. For instance, a rectangle core pattern is generated for the leftmost feature, and sidewall patterns are deposited around the four sides of this core pattern. Similarly, sidewall patterns are generated for the rest of the core patterns as shown in Figure 2.2(c). These sidewall patterns protect the underlying layer from being etched in the trim mask. Next, the core patterns will be removed (Figure 2.2(d)), leaving the sidewall patterns on the layout area.

Finally, trim mask is used to trim out the desired features as shown in Figure 2.2(e). Again, only the area that is covered by trim patterns but not covered by sidewalls will be etched. The final result shows that the features can be printed exactly even if two masks are misaligned, as long as the amount is small enough to be bounded by the sidewalls. Clearly, the sidewalls



(a) Original features

(b) Core patterns

(c) Sidewall deposition

(d) Removal of core patterns

(e) Applying trim mask to produce target features

(f) Final patterns

Figure 2.2: Illustration of SADP process.

in SADP process help to tolerate the unexpected process variation when printing two masks. By carefully placing core and trim patterns, overlay can be avoided in SADP. Figure 2.2(f) shows the printed patterns, which are located exactly at the desired places.

The above example illustrates how SADP works and how it can be used to avoid overlay. Various works [22]–[25] have demonstrated the advantages of SADP in the sub-30 nm process. Note that without further notice, we will continue to use the same color coding above in the remaining chapter.

2.1.2 SADP Decomposition Problem

To fully explore the potential of SADP, we have to address the following problem. Given a layout, find a decomposition that consists of a set of core patterns and trim patterns that can produce the layout exactly without overlay after SADP process. Note that there may exist many different decompositions for a given layout, while sometimes there may exist no solution. The above formulation has several problems:

1. The no-overlay constraint may be so tight that it may be very hard to obtain a decomposition for a layout, especially when the layout is not optimized for SADP process. Conversely, it is also very challenging to design a layout that satisfies the no-overlay requirement.

2. Some overlay may be tolerated to a certain extent. For example, we may allow reasonable overlay to appear at the metal wire endings, which will not pose a serious problem as long as it does not cause incorrectly connected metals.

3. The designer should be able to decide the trade-off and risk by having the flexibility to specify feature edges where overlay may happen.

Thus, instead of not allowing any overlay completely, we should have the flexibility to allow overlay at specified locations. Specifically, we should always disallow overlay in important feature edges (critical edges) as the circuit function may be impacted if overlay happens at these edges. Meanwhile, we allow overlay in those non-critical feature edges when no serious problem will occur even if overlay happens. We define the *SADP decomposition problem* as follows:

Definition 1 (SADP Decomposition)**.** *Given a layout, find a decomposition that consists of a set of core patterns and trim patterns that can produce the layout exactly without overlay at critical edges after SADP process.*

Such a decomposition is referred as an overlay-free or a no-overlay decomposition. Note that the term no-overlay is always used with respect to critical edges. We are interested in finding such decompositions efficiently as:

1. Overlay at critical edges may impact the circuit function. We should maximize the advantage of SADP to eliminate the overlay. Meanwhile, while we should allow a certain overlay such that we have more flexibility in designing the layout.

2. Designers may want to check if a given design is SADP-compliant, particularly in an interactive (real-time) fashion.

2.1.3 Our Contributions

The SADP decomposition problem described above is indeed very difficult, because due to the technology constraints, two core (trim) patterns cannot be printed simultaneously if their distance is too small (minimum distance design rule). One possible approach to the problem is to use a graph formulation similar to the one used in LELE [26]. A pair of features that cannot be printed simultaneously in the core mask can be viewed as a pair in conflict. Hence, it is natural to construct a *constraint graph* according to the conflicts, and use a graph two-coloring algorithm to find out the set of features that can be printed simultaneously in the core mask. However, the above graph formulation cannot capture some cases in SADP. As we will show later, the two-colorability of this graph is only a *necessary condition* for the SADP decomposition problem (instead of necessary and sufficient for the LELE case). Further, it is non-trivial to find a decomposition from the two-coloring solutions, since a two-coloring solution only tells us what features can be assigned core patterns for simultaneous printing. For the other features, we still need to introduce core patterns and generate the sidewalls that can define them. Moreover, there may be many connected components in a graph and the number of two-coloring solutions is *exponential*. Not every one of them can be translated to a feasible decomposition.

In fact, the general SADP decomposition problem is proven to be NP-complete [27]. However, we showed that if we disallow overlay at critical edges (and thus disallowing the *merging technique*, which will be introduced later), the problem is tractable. In this work, we propose that such an algorithm can solve the above problem exactly in polynomial-time.

The major contributions of this chapter are summarized as the following:

- We present the first work on the SADP decomposition problem that considers critical edges. To the best of our knowledge, this is the first work that addresses the SADP decomposition problem with consideration of critical edges, where other works tackle the decomposition problem by minimizing the overall overlay and/or without considering the critical edge information.
- We propose a graph formulation for the SADP decomposition problem. We show that the two-colorability of such a graph is only a necessary condition for the SADP decomposition problem.
- We propose a polynomial-time exact algorithm that solves the above SADP decomposition. Our algorithm computes the decompositions for each graph component, and finds a final decomposition in polynomial-time with a 2-SAT formulation.

The rest of the chapter is organized as follows. Section 2.2 gives a detailed introduction to overlay issues and design rules in SADP. We discuss some of the prior work on the SADP layout decomposition in Section 2.3. The details of our algorithm for general 2D layout will be covered in Section 2.4. Section 2.5 shows the experimental results of the proposed algorithm. Finally, conclusions will be drawn in Section 2.6. The work in this chapter was first published in [28], then extended in [29].

2.2 Preliminaries

2.2.1 Overlay in SADP Process

We refer to the maximum possible overlay of the trim mask as *trim mask overlay*, and denote it as w_o. Overlay control in SADP is achieved by pro-

viding sidewall protection to feature boundaries. We denote the thickness of the sidewall as w_s. Figure 2.3 provides a detailed scenario of printing the left-most feature in Figure 2.2. The feature boundaries are protected by the sidewall, and the boundaries of the trim pattern are located inside the sidewall. The trim mask may shift left or right within the distance of w_o during the SADP process. However, it is still inside the sidewall and will not generate incorrect features. Ensuring feature boundaries to be protected by the sidewall is the key to avoiding feature overlay. In particular, we observe that the sidewall must have thickness $w_s \geq 2w_o$ to fully tolerate the overlay. Furthermore, the trim pattern must be placed such that it is contained in the sidewall after the trim process. In other words, the trim pattern must be at least w_o away from the two sides of the sidewall to avoid feature overlay.

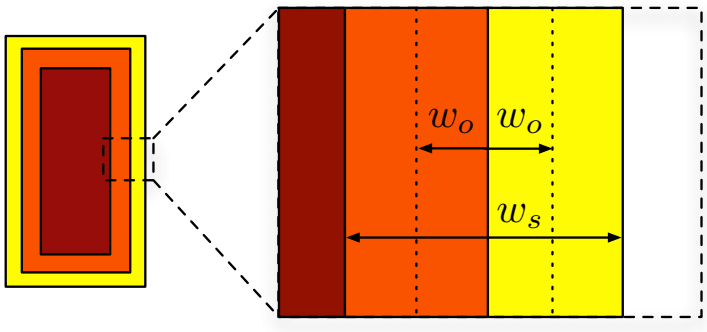


Figure 2.3: Trim mask overlay and sidewall thickness.

2.2.2 Process Rules

Design rules are always necessary for SADP-compliant manufacturing. We will adopt the following process rules in this chapter:

- The width of a core pattern is at least w_c.
- The width of a trim pattern is at least w_t.
- The distance between core patterns is at least d_c.
- The distance between trim patterns is at least d_t.
- The width of sidewall is w_s.

- The maximum trim overlay is w_o.

In practice, w_c and w_t are usually the same; without further notice we refer to them as $w_{\min}$. d_c and d_t should also be the same, and we refer to them as $d_{\min}$. From the discussion in the previous subsection, we know that $w_s \geq 2 \cdot w_o$ must be satisfied in order to guarantee the existence of a decomposition. In our work, we assume that the original features are all rectilinear polygons and all the features have widths larger than the minimum width of the mask.

2.3 Previous Work

As of the publication of this work, all of the previous works addressed the problem indirectly by minimizing the overall overlay, quantified by the length of the feature edges that are not defined by sidewall. Zhang *et al.* proposed to solve the problem with a satisfiability (SAT) formulation in [30]. In this approach, the layout area is divided into grids. A 0-1 binary variable is created to denote whether the grid is assigned a core pattern or not. Then, the design rules and geometry constraints are formulated as Boolean expressions in SAT. The objective is to determine whether there is a satisfiable assignment. However, modeling geometry constraints with Boolean expressions, such as 'rectilinear shape', will generate a huge amount of Boolean expressions. Furthermore, the number of variables depends on the granularity of the grids. As a result, the number of variables will blow up when a fine grid is used, which is not scalable.

An integer linear programming (ILP) formulation problem is proposed in [2], where the constraints were expressed in linear inequalities, and the objective is to minimize the total overlay. This approach suffers from a similar drawback as the SAT approach. Moreover, the objective may not directly reflect the solution quality, as the amount of overlay is not directly related to the yield. For instance, a small amount of overlay at a critical edge is already enough to cause a problem, while lots of overlay at non-critical edges may be allowable. Finally, both SAT and ILP are NP-complete in general, and thus these methods cannot scale to large problems.

Ban *et al.* presented a graph-coloring based heuristic in [3], [4]. The edge weight corresponds to the parallel run length of the feature edges that are violating the process rule. A two-coloring is then constructed heuristically

to minimize the total weight of the edges where the incident vertices are the same color. Note that this is not a valid two-coloring solution in the traditional sense, since two adjacent vertices can have the same colors (and thus conflict).

To resolve the coloring conflict mentioned above, *i.e.*, a feature pair that violates design rules, the authors used a 'merging' technique, where the two conflicting features will be merged as one, and they will be trimmed out later using two separate trim patterns. An example cited from [4] is shown in Figure 2.4. The features B, C and D form an odd cycle (Coloring conflict) and thus there is no two-coloring solution. The merging technique will treat features B and C as a single feature F (Grouping and Merging). Finally, a trim pattern that has an opening between the two conflicting boundaries of B and C will be used to trim them out from the merged pattern. Note that the red mask pattern in 'Trimming' denotes the *open* of trim pattern, *i.e.*, the trim pattern should be its complement.

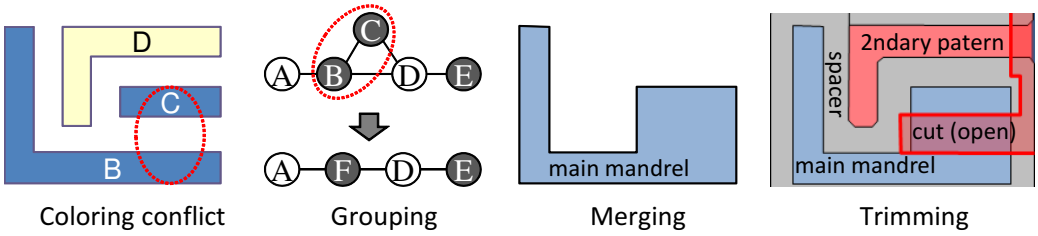


Figure 2.4: Merging technique used in [4].

The merging technique is currently used in industry, which adds flexibility in SADP [20]. However, whenever it is used, the merged features must be cut out using trim mask. Thus, the edges where the merging occur will be defined by trim mask and thus will suffer from potential overlay. In this work, we will focus on the problem without merging such that we can find a no-overlay decomposition if one exists.

2.4 A Polynomial-time Exact Algorithm for SADP Decomposition

In this section, we first revisit and illustrate the concept and usage of auxiliary cores, then discuss the effect of critical edges and how it will affect

the solution space of the decomposition problem. Next, we carefully examine how the design rules will impact core and trim patterns. Based on the above observations, we propose a concept *SW-graph* that captures the design rules, and show that the two-colorability of the SW-graph is only a necessary condition for SADP layout decomposability. Finally, we present an exact algorithm that finds a decomposition in polynomial-time. At the end of the section, we also provide a brief analysis of the correctness and complexity of our algorithm.

2.4.1 Auxiliary Cores

As we mentioned in the previous sections, we can use *auxiliary core* to help generate sidewalls for the features that do not have a corresponding core pattern. More formally, an auxiliary core is a core pattern that is placed outside of a feature to generate the required sidewalls to define some critical edges. To distinguish, we refer to a core pattern as a *main core* when it has the same shape as a feature and is placed at the same location of this feature to generate the required sidewalls.

We have already seen that one way to print a feature is to use a main core as shown in Figure 2.3. However, we may not be always able to use this approach due to the process rule constraint. An alternative way is to use auxiliary cores. Observe that to print a feature exactly without overlay at any edges, there must be a *ring of sidewall* surrounding the feature as shown in Figure 2.5(a). More generally, a set of *required sidewalls* must be generated for the critical edges of a feature. The ring of sidewall is a special case where all the edges are critical. For the purpose of illustration, we assume in this example that all of the edges of the feature are critical. Besides using a main core to generate these sidewalls, we can use one of many auxiliary cores around the feature to generate the ring of sidewall. The complete decomposition that includes sidewall and trim pattern is shown in Figure 2.5(b). Note that from now on we include the core mask, sidewalls and trim mask into one figure for compactness. This example demonstrates how to print a feature using auxiliary core(s) instead of a main core. We refer the approaches used in Figure 2.3 (placing a core pattern at the feature location) and Figure 2.5(b) (placing an auxiliary core outside to generate the

sidewalls) as main-method and aux-method.

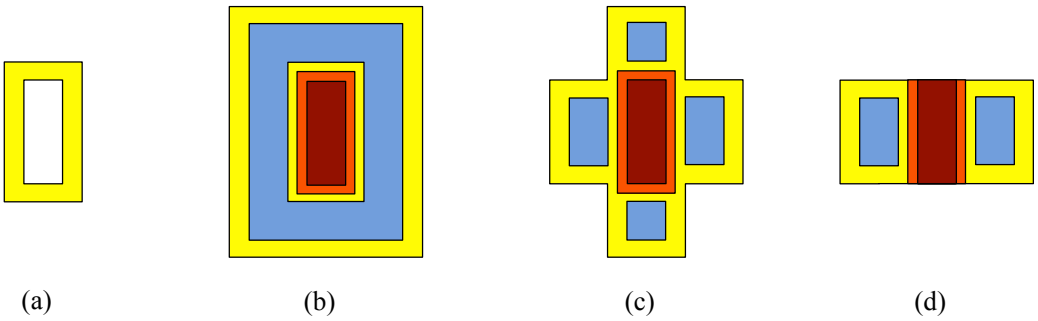
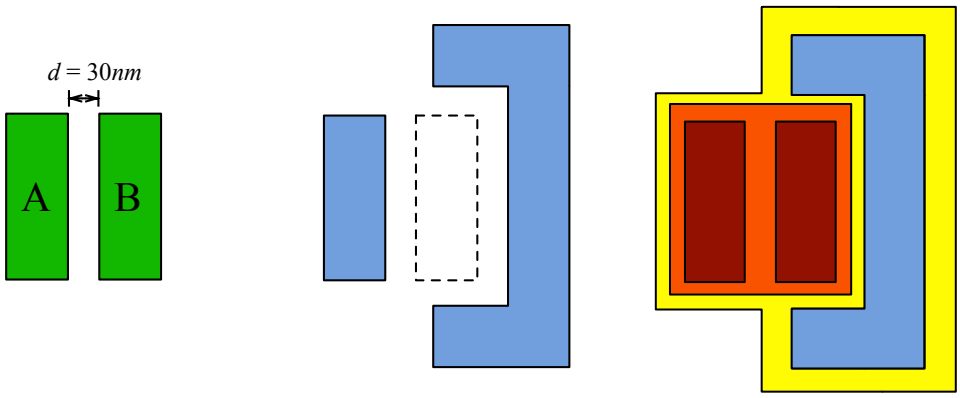


Figure 2.5: (a) Ring of sidewall. (b) Core, trim mask and sidewall. (c) Alternate decomposition. (d) Critical edges example.

Using aux-method, we can obtain different decompositions. For instance, Figure 2.5(c) illustrates another possible decomposition that uses four auxiliary cores to generate the required sidewalls. Clearly, this is also a valid decomposition that will not have overlay. For the case that not all the edges are critical, we can use separate auxiliary cores to generate sidewalls. For instance, if the top and bottom edges of the feature are non-critical, we can use the decomposition shown in Figure 2.5(d) to generate the sidewalls of its left and right sides.

The above example shows the basic usage of auxiliary cores. They become extremely important when there are multiple features. A more sophisticated example is illustrated in Figure 2.6. In Figure 2.6(a), the two features are too close to be printed simultaneously in core mask. Figure 2.6(b) shows how we can obtain an overlay-free solution by using auxiliary cores. If we use a main-core to feature A, the sidewall generated can then protect the left edge of feature B. Then, to avoid overlay for the other three edges of feature B, we use an auxiliary core that can generate the remaining required sidewalls. The final result is shown in Figure 2.6(c). We use a single trim pattern to trim out both features. Clearly, this decomposition has a symmetric solution where a core pattern is used for feature B, and an auxiliary core to generate a sidewall for feature A. This again shows the flexibility of SADP decomposition. Nevertheless, the distance between an auxiliary core and the feature that is being assisted must satisfy some requirements. This is discussed in the following section.



(a) Target feature (b) Core mask (c) Sidewall and trim mask

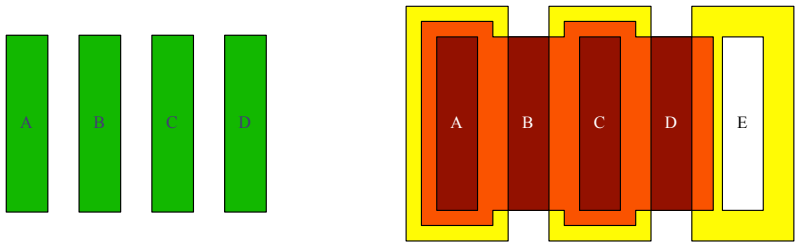
Figure 2.6: Using auxiliary cores to produce target features.

2.4.2 Critical Edges

We now discuss the effect of critical edges. As we mentioned before, critical edges are the feature edges that should not have any overlay. However, if we disallow overlay anywhere, *e.g.*, all the feature edges are critical, it may be very hard to find a valid decomposition even for a simple layout. We illustrate this by showing an example.

Figure 2.7 shows a layout that contains four features of line shape, where each of them has a width $w_{\min}$. The distance between adjacent features is w_s. We further assume the amount of overlay is much smaller than the sidewall width. Clearly, we can only simultaneously print the features in an alternating fashion, *i.e.*, either use core patterns in features A and C or features B and D. In either way, there are always two features that have no sidewall protection. Let us assume core patterns are used for features A and C. We further place an auxiliary core pattern E to the right of feature D to generate sidewall for D's right side. Now, one may want to use auxiliary cores to generate sidewalls for the unprotected top and bottom tips of features B and D. However, because of the existence of core patterns in features A and C, no valid core patterns can be put to generate these sidewalls. Thus, there is no way to decompose this layout without overlay. However, if these tips are marked as 'non-critical', we can safely use the decomposition as shown in Figure 2.7(b), where the core patterns are placed on A, C and E and removed. This example shows that with critical edges correctly identified, we may decompose an otherwise undecomposable layout using SADP process.

In practice, usually the feature side is critical as they affect the width of the metal wire, and thus the resistance, conductance, etc., while the feature tips are less critical as long as the effect is minor.



(a) (b)

Figure 2.7: Critical edges example.

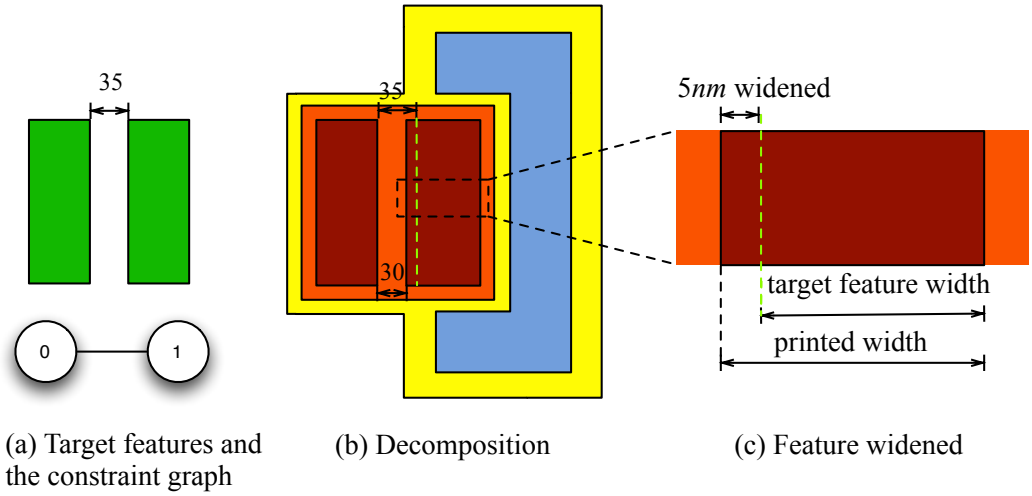
2.4.3 A Graph Formulation for SADP Decomposition

In this section, we examine the impact of the distance between features. We define the distance d between two features as the distance of two closest edges between two features. We will first show that the two-coloring solution in the conflict graph used in LELE is not necessarily converted to a no-overlay SADP decomposition. In fact, there are more cases to consider for the distance between features than the criteria used in LELE. For better illustration, we assume the $d_{\min} = w_{\min} = 40$ nm, $w_s = 30$ nm and $w_o = 10$ nm in the following.

Clearly, when the distance between two features is greater than $d_{\min}$, we can use main-method to print them simultaneously. When the distance is smaller than w_s and at least one of the closest edges is critical, there is no way to print them because the width is smaller than the sidewall width.

Now consider the two target features in Figure 2.8(a), where the distance between them is 35nm and both edges are critical. According to the constraint graph formulation in LELE, a graph with two vertices will be constructed. Without loss of generality, assume that the left feature is assigned a core pattern in the coloring solution. We cannot use two separate trim patterns to trim out the features, for otherwise they will be too close and violate the process rules. Thus, we can only use a single trim pattern for both features as shown in Figure 2.8(b). However, the sidewall generated by

the left feature cannot touch the right feature since $w_s = 30$ nm. As a result, after trimming the right feature generated will be widened 5 nm as shown in Figure 2.8(c). This overlay may severely affect the design. In conclusion, there is no way to decompose this layout and print the features exactly. The conflict graph formulation cannot capture this correctly.



(a) Target features and the constraint graph (b) Decomposition (c) Feature widened

Figure 2.8: Example of widened feature.

Another issue we need to consider is how we place the trim patterns. For a single feature, the trim patterns can be viewed as a polygon that has enlarged shape of the original feature. Given two features, due to the minimum distance rule, we have the following cases depending on the criticality of the closest edge pair and their distance d:

- If both edges are critical, then d must satisfy $d \geq 2w_o + w_{\min}$ to put the trim patterns.
- If only one of the edge is critical, then d must satisfy $d \geq w_o + w_{\min}$ to put the trim patterns, where the one edge of the trim pattern overlaps with the non-critical edge.
- If both edges are non-critical, then d must satisfy $d \geq w_{\min}$, where either trim pattern will have one edge overlap with the non-critical edge.

In conclusion, the distance between the features must be carefully considered and reflected in the constraint graph. Given two features in Figure 2.6(a), we will have the following cases according to the criticality of feature edges and distance d:

- If $d < w_s$, this layout cannot be decomposed because the generated sidewall will overlap with at least one feature.
- If $d = w_s$, we can use main-method for one feature and use aux-method for another. The generated sidewall will protect both feature edges.
- If $w_s < d < w_{\min}$, This layout cannot be decomposed, because they are so close that we cannot put two separate trim patterns to cut them out, nor can we use a merged trim pattern because there is no way to generate sidewall for both edges.
- If $w_{\min} \leq d < w_o + w_{\min}$,
  - If at least one edge is critical, we should use main-method for the features. The trim patterns need to be merged.
  - If none of the edges is critical, we can use aux-method for the features. The trim patterns can be separated.
- If $w_o + w_{\min} \leq d < 2w_o + w_{\min}$,
  - If both edges are critical, we should use main-method for the features. The trim patterns need to be merged.
  - Otherwise, we can use aux-method for the features that have non-critical edge. The trim patterns can be separated.
- If $2w_o + w_{\min} \leq d < 2w_s + w_{\min}$,
  - If both edges are critical, we should use main-method for the features. The trim patterns need to be separated.
  - Otherwise, we can use aux-method for the features that have non-critical edge. The trim patterns can be separated.
- If $d \geq 2w_s + w_{\min}$, the features are far away enough from each other. We can use either method to print the features separately.

Clearly, the distance between features is closely related with w_s, *i.e.*, the width of the sidewall, and the criticality of the closest edges. Given a layout that consists of a set of target features, we construct $G_s = (V, E)$ as follows:

- For each feature in the layout, include a vertex in V.

- Insert an edge $e = (u, v)$ between two vertices u and v if $d = w_s$.

Meanwhile, the layout is not decomposable if $d < w_s$ or $w_s < d < w_{\min}$.

We refer to the above graph as an *SW-graph*. We can use two-coloring to find a set of core patterns that can be printed simultaneously in core mask. In the following, we will refer to the two colors as 'core' and 'space'. For the feature that is assigned a 'core' color, we put a core pattern at its location. The following theorem shows the relationship between the two-colorability of G_s and SADP decomposition problem:

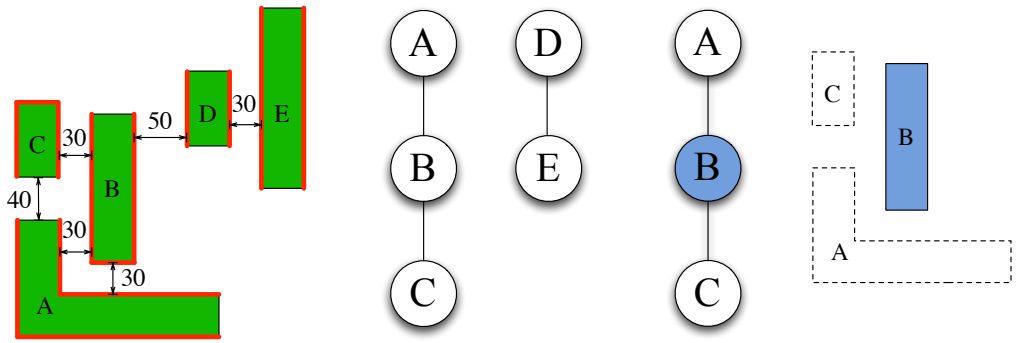
Theorem 1. *Two-colorability of G_s is a necessary condition for SADP layout decomposability.*

Proof. If the feature distances does not satisfy the conditions we discussed above, we know the layout cannot be decomposed. Now, we assume the feature distances satisfy the condition. Given a decomposition of the layout, we simply color a vertex as 'core' when a core is assigned to its corresponding feature, and the remaining vertices as 'space'. For each $e = (u, v)$ in G_s, u and v must be colored differently, otherwise the decomposition is invalid. Hence, we obtain a two-coloring solution from the decomposition. As result, two-colorability of G_s is a necessary condition of whether the layout is SADP-decomposable. □

Clearly, building the graph and finding two-coloring solutions for a graph can be done efficiently in polynomial-time. As a side note, a graph is two-colorable if and only if there exists no odd cycle in the graph. An example of the graph formulation is shown in Figure 2.9. Figure 2.9(a) shows a layout with five features. The critical edges are marked in red. Figure 2.9(b) shows the SW-graph of this layout. Figure 2.9(c) shows one of the two-coloring solutions for the connected component {A, B, C}. This example will be used throughout the chapter to illustrate the proposed algorithm.

2.4.4 Overview of Our Algorithm

Our algorithm contains two stages. In the first stage, the algorithm finds decompositions for each of the connected components in the SW-graph separately. We will introduce a method to find the decompositions according to



(a) Layout (b) SW-graph (c) Two-coloring of {A, B, C}.

Figure 2.9: Layout example and its SW-graph. The blue color denotes a 'core'.

the two-coloring solutions. Despite the fact that we may obtain many legal decompositions from the two-coloring solutions of a connected component, we will show that there are in fact at most two special decompositions with the property that all the other decompositions are a subset of either one of them.

In the second stage, a set of decompositions is selected from the connected component decompositions and combined as a final complete decomposition. In particular, exactly one decomposition will be selected for a connected component in such a way that their combination is still an overlay-free decomposition. In other words, the core and trim patterns in the resulting decomposition will not violate process rules or cause overlay. Although there may be exponentially many combinations, we show that the problem of choosing compatible decompositions can be reduced to a 2SAT (2-Satisfiability) problem and thus can be solved efficiently in polynomial-time.

2.4.5 Decompositions for Connected Components

In the previous section, we conclude that the two-colorability of the SW-graph is only a necessary condition for the problem. From a two-coloring solution, we can only obtain a set of non-conflicting features that can be printed simultaneously using SADP process. The remaining problem is to find a set of auxiliary core patterns that can generate the required sidewalls to define the features. Our algorithm divides this problem into two stages. This section describes the first stage, which finds the decomposition for the

features in a connected component of SW-graph separately.

Let us consider a connected component in G_s and its two-coloring solutions. Based on the two-coloring solution, we assign a main core to a feature that is colored as 'core'. A trim pattern that has the same shape as the feature is also assigned and scaled in such a way that it has a distance w_o away from the feature edges. By doing the scaling, we can make sure the trim pattern will be contained in the sidewalls even if misalignment happens.

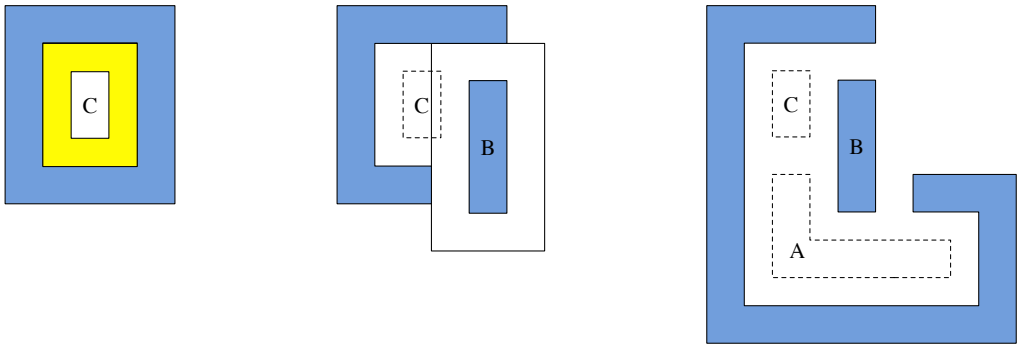
For each feature that is assigned a 'space', we generate a *ring of auxiliary core* with width w_c placed outside of the feature to generate the required sidewall. Figure 2.10(a) illustrates the ring of auxiliary core of feature C in Figure 2.9. Note that the ring of core may 'interact' with other core patterns and features. This can be addressed according to the following procedure:

1. The ring of auxiliary core conflicts with some main cores or other features in the same component. Since the positions of the main cores and features are fixed, the only solution is to trim the conflicting parts from this ring. An example is shown in Figure 2.10(b), where feature B forces part of the auxiliary core for C to be removed. Note that the solid line around feature B denotes the area that no other core pattern can be inside. It has a width $w_c = 40nm$ in our case.

2. The ring of auxiliary core overlaps with some other auxiliary cores. In this case, we can safely *merge* them together. Note that the merging at this step is to combine auxiliary cores together. Any design rule violation will be fixed during post-processing.

3. After the trimming or merging, if the shape of the auxiliary core violates the process rule, we *post-process* it to satisfy the rules. For example, we remove the part that violates the minimum core width rule. Figure 2.10(c) shows the auxiliary cores for feature A and C merged together, with some parts post-processed.

In the above procedure, we start with a ring of auxiliary core regardless of the criticality of edges. The reason is that although we do not require a non-critical edge to have sidewall protection, it is still useful when it is not causing conflict. If the auxiliary core that generates the required sidewall for a non-critical edge is causing conflict, it can always be removed.



(a) Ring of core for feature C

(b) Removal of conflicting part

(c) Merging auxiliary cores

Figure 2.10: Ring of core, removal, merging and post-processing.

After the above procedure, some design rules may be violated after trimming or merging the auxiliary cores. The post-processing step will try to fix these violations, including distance rule violation and minimum width rule violation. To handle distance rule violation, we will try to insert an auxiliary core between the two auxiliary cores that cause this violation. In particular, we push the two conflicting edges toward each other until they meet. This may cause some other conflict, and the conflicting part should be removed. To handle minimum width rule violation, a similar operation is performed. We push the two edges at the 'bottleneck' part that causes the conflict until it satisfies the width rule. After the post-processing, if the violation cannot be fixed, we will report no feasible decomposition.

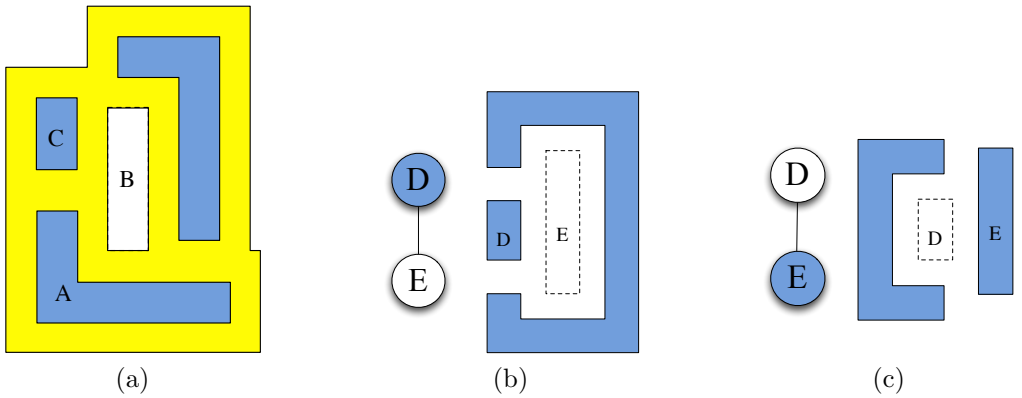
After the above process, if the required sidewalls cannot be generated by the auxiliary cores, we conclude that there is no corresponding decomposition for this two-coloring solution. This is due to the removal of auxiliary cores that leads to failure of generating the required sidewall. As a result, the whole layout will not have a decomposition. Note that if the lengths of the remaining auxiliary cores do not satisfy the minimum width rule, we should not remove it in the above procedure. The reason is that it may become legal after merging with other auxiliary cores. This will be discussed in detail in the second part of the algorithm.

As we have seen before, there may be various sets of auxiliary cores that can generate the ring of sidewall for a given feature, and thus there may be many different decompositions. However, we should use the auxiliary core found by the above approach. To see this, we first assume that the auxiliary cores of

minimum width are used. Clearly, a sidewall pattern can only be generated by an adjacent core pattern, and the ring of auxiliary core is defined using the ring of sidewall. In other words, the ring of auxiliary core is actually the union of all possible decompositions, because the sidewall it generates will contain the sidewalls that are generated by all possible decomposition. If the auxiliary core after the above process cannot provide required sidewalls, neither will other auxiliary core patterns.

Note that in the above discussion, we have assumed that the width of the auxiliary cores are w_c. Using the minimum width is already enough to satisfy the process rules. Any set of auxiliary cores using a larger width is just a superset of the set of auxiliary cores that have minimum widths. If the auxiliary core with minimum width does not exist, a wider one will not exist, either.

In summary, the decomposition derived using the proposed approach, *i.e.*, using ring of auxiliary core and post-processing, contains all the possible decompositions that correspond to a two-coloring solution. In the following, we will refer to such a decomposition as a *standard decomposition*. Figure 2.11(a) shows the decomposition obtained from another two-coloring solution. Figure 2.11(b) and 2.11(c) show the standard decompositions for the two two-coloring solutions of connected component {D, E}. For simplicity, only the core patterns are shown.



(a) (b) (c)

Figure 2.11: Standard decompositions.

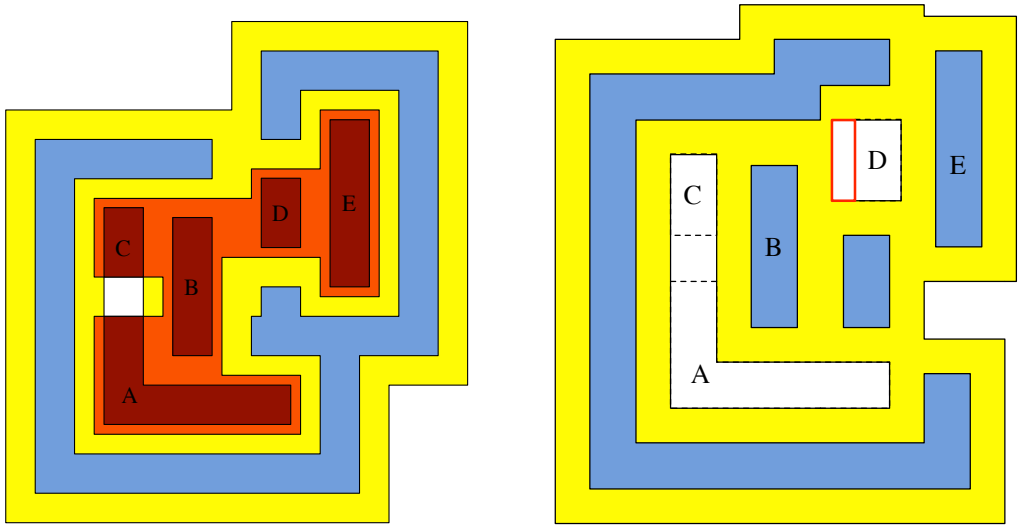
2.4.6 Combining Decompositions

From the above analysis, we know that there will be at most two standard decompositions for a connected component, since there are exactly two two-coloring solutions and it is possible that no decomposition can be found for a two-coloring solution. The remaining problem is how we can combine the standard decompositions and form a complete decomposition to the original layout.

Observe that the 'interaction' between two decompositions will only be caused by the auxiliary cores; *i.e.*, the auxiliary cores from one decomposition may 1) conflict with the main cores or features in another decomposition, or 2) overlap with some auxiliary cores in another decomposition. Hence, the 'interaction' between two decompositions is the same as what we have discussed in the first stage. We can thus apply the same process to obtain a combined decomposition. Similar arguments suggest that if some auxiliary cores in the combined decomposition do not satisfy the minimum width constraint, a combined overlay-free decomposition cannot be found from the two decompositions.

We now define *compatibility* between two decompositions from two different components. We refer two decompositions as *compatible* if they can be merged as an overlay-free decomposition. Otherwise, they are *incompatible*. As an example, Figure 2.12(a) and Figure 2.12(b) show the two combinations of the decompositions from two connected components, where the first pair is compatible (and also a complete decomposition for the layout) and the second incompatible. For the compatible decomposition, we also show the trim pattern that can print the features exactly in the figure. Note that since the bottom of feature C and the top of feature A are non-critical, the sidewall is not needed between them and the trim pattern is used to defined these tips.

Our objective is to include exactly one decomposition from each graph component to form a complete decomposition. All the decompositions selected should be mutually compatible. In Section 2.1, we have shown that the number of final decompositions may be huge. Because there are at most two decompositions in a component, and the number of components may be up to the number of features n, the number of final decompositions will be in $O(2^n)$. However, this problem can be solved efficiently, as the following



(a) Compatible decomposition (b) Incompatible decomposition

Figure 2.12: Combining decompositions of {A, B, C} and {D, E}.

theorem indicates.

Theorem 2. *Given decompositions of all components of G_s, a complete decomposition can be found in polynomial-time.*

Proof. We show this by reducing the problem to 2SAT (2-Satisfiability), which is a special case of the general Boolean Satisfiability problem. 2SAT asks to determine whether a collection of binary variables with paired constraints can be assigned values satisfying all the constraints.

Assume that there are k components in G_s, each with at most two decompositions. We construct a Boolean formula f as follows. For component i, we denote its two decompositions as x_i and $\bar{x}_i$. If x_i ($\bar{x}_i$) does not exist, include its negation $\bar{x}_i$ (x_i) in f as a conjunction. For each decomposition, we check whether it is compatible with all the decompositions except the one in the same component. Hence, there will be at most $O(n^2)$ incompatible pairs. Conjunct a clause $\overline{(x_i \wedge x_j)}$ in f if x_i is incompatible with x_j where $i \neq j$. Clearly, f is in *2-conjunctive normal form* (2CNF).

We show that f is satisfiable if and only if there exists a complete decomposition; the satisfiable assignment for f is the complete decomposition. The see this, notice that if there is a satisfiable assignment for f, all the clauses in f must be true. In other words, no incompatible pairs will appear in the clause, and thus the complete decomposition contains mutually

compatible decompositions only. Conversely, if we have a complete decomposition, where all sub-decompositions in it are compatible, no incompatible pair exists and thus we have a satisfiable assignment for f. Finally, there exist efficient algorithms for 2SAT problem such as [31]. □

The correctness and complexity of our algorithm are concluded in the following theorem.

Theorem 3. *The proposed algorithm solves the SADP layout decomposition problem exactly in polynomial time.*

Proof. In the first stage of the algorithm, we are generating ring of auxiliary cores in the decomposition, which are essentially the combination of all candidate decompositions. We only remove the minimal parts that violate the process rules. When the remaining auxiliary cores do not satisfy the process rules, it means all of the candidate decompositions are removed due to conflicts. Hence, if there exists a feasible decomposition, it must be contained in the decompositions found so far. In the second stage, the way we define compatible decompositions is the same as in the previous stage. Hence, the consequence is the same, *i.e.*, we will not eliminate a feasible decomposition during the procedure. Finally, the set of compatible decompositions is a feasible complete decomposition of the layout since it does not violate process rules anywhere. Hence, the proposed algorithm finds a decomposition if one exists and is thus exact.

The complexity of our algorithm is presented as follows. Checking the distance of features is essentially a collision detection problem, which can be done in $O(n^2)$ using existing computational geometry techniques. Constructing the graph and performing two-coloring can be done in $O(n^2)$ and $O(n)$, respectively. In the first stage, we mainly deal with Boolean polygon operations, *e.g.*, intersecting, removing and merging the polygons. All of these can be done in polynomial-time. A similar process exists in the second stage. Furthermore, the second stage requires constructing a 2SAT formula from the component. Since the number of components is at most $O(n)$, the pairwise compatibility check requires $O(n^2)$. Finally, the 2SAT problem can be solved using Tarjan's strongly connected component based algorithm [31], which runs in linear time. In conclusion, the proposed algorithm runs in polynomial time. □

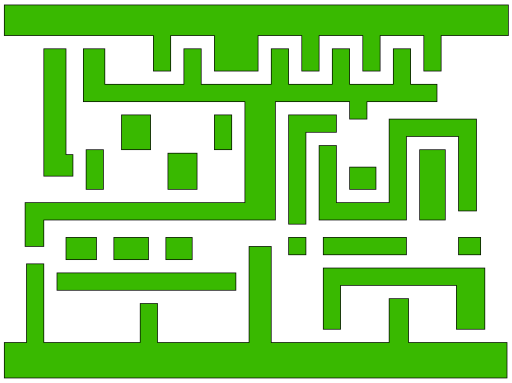
An outline of our algorithm is given in Algorithm 1. Figure 2.13 shows the decomposition of running an implementation of our algorithm on a layout with moderate size of features.

Algorithm 1 SADP Decomposition Algorithm

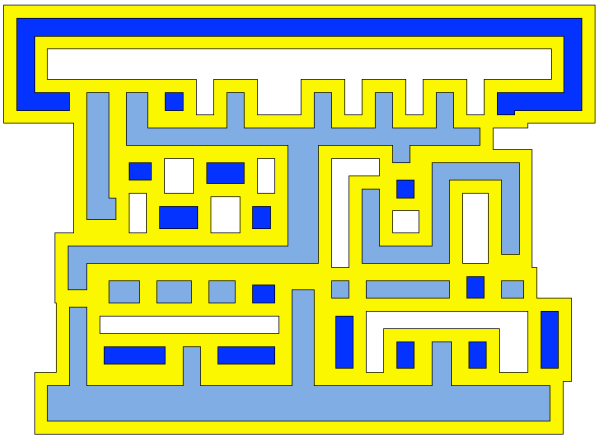
```
1: procedure DecomposeLayout(F)          ▷ F is a set of target features
2:     Construct G_s from T
3:     Report undecomposable if feature distances are infeasible
4:     C ← ∅, T ← ∅                          ▷ C: cores, T: trims
5:     for all Component g in G_s do
6:         TwoColor(g)
7:         Report undecomposable if g is not two-colorable
8:         for all Two-coloring solutions of g do
9:             for all Feature f colored as 'core' do
10:                C = C ∪ c, a main core for f
11:            end for
12:            for all Feature f colored as 'space' do
13:                C = C ∪ c, a ring of auxiliary core for f
14:                Post-process the auxiliary cores in C
15:            end for
16:            T = T ∪ t, a trim for C
17:        end for
18:    end for
19:    Post-process all the auxiliary cores and trim patterns
20:    Combine compatible decompositions as a final decomposition D
21:    return D
22: end procedure
```

2.5 Experimental Results

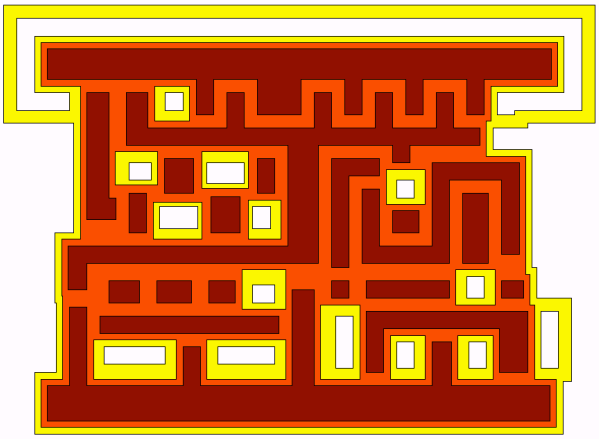
To demonstrate the efficiency of our algorithm, we implemented our algorithm in C++ programming language, and performed experiments on a 3.20 GHz Intel Xeon CPU with 32GB memory. We tested our program with the following process rules. The minimum space between the core (trim) patterns and the minimum width of the core (trim) patterns are all set to be 40 nm, while the sidewall width and trim mask overlay are set to be 30 nm and 10 nm. We used the 45 nm Nangate Open Cell Library [32] as our test data. Since the library is not designed for SADP, most of the layouts do not have an overlay-free decomposition. We scaled and adjusted the cells to conform



(a) Features



(b) Core mask and sidewalls



(c) Trim mask

Figure 2.13: Running our algorithm on general 2D layout. Note that a darker blue is used for auxiliary cores to better distinguish from main cores.

Table 2.1: Comparison Between Our Method And Ilp

Name	# Fs.	# Cs.	Runtime (s)	
			Ours	ILP
INV_X1	4	2	0.04	7.18
BUF_X1	5	1	0.04	3.45
BUF_X16	5	3	0.03	38.26
NAND2_X1	5	3	0.10	36.87
AND2_X1	6	2	0.08	25.33
AND3_X1	7	3	0.14	48.90
AND4_X1	8	4	0.32	55.90
OR2_X1	6	3	0.25	4.65
OR4_X4	8	2	0.14	5.22
XOR2_X1	7	2	0.11	4.56

Table 2.2: Experiment on Critical Edges

Name	Decomposable?	
	Original	Critical
AOI211_X2	N	Y
AOI221_X1	N	Y
OAI221_X1	N	Y
NOR2_X1	N	Y
CLKBUF_X1	N	N
DLH_X1	N	Y
SDFFS_X2	N	N

with the design rules. In particular, the features are scaled to meet the minimum width requirement 40 nm, and the odd feature distance is adjusted to either as sidewall width or minimum feature distance. We implemented the ILP method in [2] for comparison purpose, and used Gurobi Optimizer [33] as the ILP solver.

In the first experiment, all edges are critical and thus overlay should be avoided everywhere. Table 2.1 shows the performance comparison between our method and the ILP method [2], where '#Fs.' refers to number of features and '#Cs.' refers to the number of components in the SW-graph. From the table, we can see that our algorithm achieves a 166X speed-up on average compared to the ILP method.

In the second experiment, we use a set of library cells that cannot be decomposed when all edges are critical. We create another set of data based on them and specify the critical edges using the following criteria: for the

Table 2.3: Experiment on Large Benchmarks

Name	# Features	Decomposable?	Runtime
std_500_1	500	N	1.5s
std_500_2	500	Y	6.2s
std_1k_1	1000	N	3.7s
std_1k_2	1000	Y	23.4s
std_5k_1	5000	N	27.1s
std_5k_2	5000	Y	101.0s
std_10k_1	10000	N	78.9s
std_10k_2	10000	Y	257.7s
std_50k_1	50000	N	804.4s
std_50k_2	50000	Y	1244.6s

feature that has line-shape, we consider the line ends that do not have a nearby feature as non-critical. Table 2.2 shows the results of running our algorithm on the two sets of data. The columns 'original' and 'critical' denote the results on two sets of data, where the value Y/N denotes decomposable or not. We can see that for most of the previously undecomposable layouts, they can be decomposed by allowing some unimportant edges to have overlay.

In the third experiment, we created a set of large benchmarks using the cells from the standard library. In each test case, standard cells are selected randomly and concatenated to form a standard cell row. Nearby cells are randomly selected and connected using metal wires. For each feature, non-critical edges are selected with probability 10%. The test cases created in such a way can be viewed as the routing layer M1 that has complicated 2D layouts. We created test cases that contain 500, 1000, 5000, 10000 and 50000 features, respectively. For each size, we have two types of benchmarks, one can be decomposable and one cannot.

We ran the proposed method and ILP method on these benchmarks. However, the ILP method failed to decompose the layout within a reasonable amount of time (greater than 12 hours). Thus, we do not include them in the report. Table 2.3 shows the running time of our algorithm on this set of benchmarks. We can see that the proposed algorithm can solve all the test cases using a reasonable amount of time, where the largest case consumed about 20 minutes. For the cases that cannot be decomposed, the time used is far less because either the SW-graph cannot be constructed, or the decomposition cannot be found for some connected component. Thus, the program

terminates earlier than a decomposable layout.

2.6 Conclusion

In this chapter, we studied the SADP decomposition problem, where overlay must be avoided at critical edges. We proposed a graph formulation that correctly captures the design rules, and showed that the two-colorability of the proposed graph is a necessary condition for SADP layout decomposability. We presented the first polynomial-time exact algorithm that solves this problem. Experimental results demonstrate the efficiency of our algorithm. The proposed algorithm is expected to aid the designers to efficiently check whether their designs are SADP-friendly.

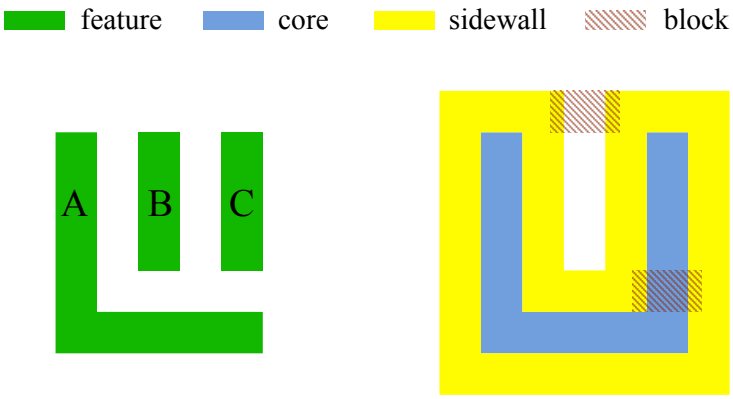
CHAPTER 3

SADP DECOMPOSITION FOR ROW-BASED STANDARD CELL LAYOUT

3.1 Introduction

Among several possible styles of SADP process, space-is-dielectric (SID) is the most widely adopted type. In the previous chapter, we introduced SADP process with core and trim mask. In this chapter, we will use the complement of trim mask, *i.e.*, block mask (or cut mask) for the purpose of clearer illustration. As we will be allowing merge-and-block technique, using block notation instead of trim notation will be much more natural.

Under this notation, the core mask will generate a set of core patterns. Spacer pattern will then be deposited surrounding the core patterns. The features are defined by the patterns that are not covered by sidewalls or block mask. Figure 3.1 shows an example of printing features using core and block mask. Note that the L-shape feature and the vertical 1D feature are merged and then cut out using a block pattern as in Figure 3.1(b). As pointed out in [20], it is desirable to avoid overlay at feature sides, *i.e.*, avoid



(a) Target layout. (b) Decomposition result.

Figure 3.1: Illustration of SID-type SADP process.

block-defined feature sides. The overlay effect at feature tips, on the other hand, is negligible.

In this chapter, we focus on the SADP decomposition problem for row-based standard cell layout. We consider the sides of the features as critical, where overlay should be disallowed. If a feature side is defined by block mask, overlay may happen and such a situation is referred to as an *overlay violation*. The objective is to minimize the total number of overlay violations. Despite the NP-hard nature of SADP decomposition in 2D layout, we show that the problem can be solved in polynomial time when the layout is row-based standard cell design. Our contribution is summarized as follows:

- We consider minimizing the total number of overlay violations in SADP decomposition as the objective. All previous works minimize the overall overlay of the layout regardless of whether the overlay is critical or not. To the best of our knowledge, this is the first work that addresses this problem.
- We propose the first polynomial-time algorithm that optimally solves the problem for a standard cell row. We further enhance the algorithm to find a global optimal solution of a multiple-row layout. Experiments demonstrate the efficiency of the proposed algorithm.

The rest of the chapter is organized as follows. In Section 3.2, preliminaries of SADP are introduced. The details of our algorithm will be discussed in Section 3.3. Section 3.4 shows the experimental results for the proposed method. A conclusion is drawn in Section 3.5. This work is published in [34].

3.2 Preliminaries

Before we discuss the proposed decomposition algorithm, we will first introduce overlay violation, row-based standard cell layout, SADP mask rules and SADP-compliant design rules used throughout the chapter.

3.2.1 Overlay Violation

We define an *overlay violation* to be the critical feature side that is not protected by sidewall (and thus defined by block mask) after the SADP

process. Figure 3.2 shows a layout and its SADP decomposition. A U-shape core pattern generates sidewalls to define the left side of feature A, right side of feature B, and bottom tips of both A and B. A T-shape block pattern defines the right side of feature A, left side of feature B and top tips of both of them. Therefore, there are two overlay violations in the decomposition; one happens at the right side of feature A, and the other one happens at the left side of feature B. Note that we consider the tips (line ends) of the features as non-critical, while the long sides are critical. Thus, the top tips of A and B are not considered as overlay violations even though they are defined by block mask.

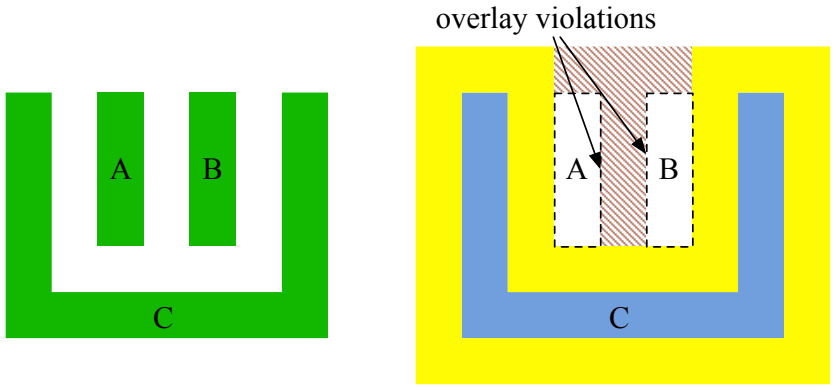


Figure 3.2: Overlay violations.

3.2.2 SADP Decomposition in Row-based Standard Cell Layout

In standard cell based designs, a library that contains pre-designed standard cells is provided to the designers. A *standard cell* is a basic logic gate (*e.g.*, a 2-input NAND gate) or small logic element (*e.g.*, a flip-flop). The standard cells in a library have the same height, but may have different widths. The designers can then use the cells to construct standard cell rows, where the cells are placed one after another in a row. Wires are routed between cells to connect the inputs and outputs. A VDD and a GND tracks will be placed horizontally at the top and bottom of each row to provide power access. Multiple standard cell rows are stacked vertically as a *row-based standard*

cell layout. A standard cell example is shown in Figure 3.3(a), and standard cell row is shown in Figure 3.3(b).

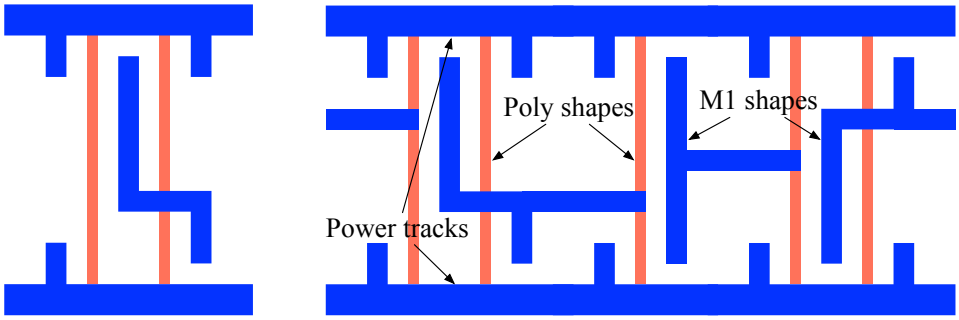


Figure 3.3: (a) A standard cell. (b) A standard cell row. Only the poly layer and the M1 layer are shown.

A standard cell layout may consist of several layers, where some of them may have preferred routing directions. A common setting is: Metal 2, 4, ... layers are vertical-preferred, Metal 3, 5, ... layers are horizontal-preferred, while Metal 1 (M1) layer is usually used for inter-cell and intra-cell routing and does not have a preferred direction, and thus the routing pattern can be dense 2D. Thus, the most difficult problem instances of layout decomposition problems usually come from the M1 layer. In this chapter, we target the decomposition problem in the M1 layer. We further formulate the problem as follows:

Definition 2 (SADP Decomposition for Row-based Standard Cell Layout)**.** *Given a row-based standard cell layout, decompose the layout into a set of core patterns and block patterns for SADP patterning and minimize the number of overlay violations.*

In the following, we refer to such a set of core patterns and block patterns as a feasible SADP decomposition.

Surprisingly, the structure of the row-based standard cell layout has interesting characteristics that can be utilized to solve related problems more efficiently than their general version (e.g., [35]). In this chapter, we will demonstrate a decomposition algorithm that utilizes the fixed width and row-structure.

3.2.3 SADP Mask Rules

We use the following mask rules for SADP:

1. The minimum width of a core (block) pattern is d.
2. The minimum distance between two adjacent patterns is s.
3. The width of sidewalls is w.

In practice, we have the following constraints [27]:

$$w = d \tag{3.1}$$

$$d < s \tag{3.2}$$

$$s < d + 2w \tag{3.3}$$

3.2.4 SADP-compliant Design

Due to the distinguished property of SADP process, some rules must be assumed to avoid potential MRC (manufacturing rule checks) errors. As pointed out by Ma *et al.* in [20], the MRC challenges mostly come from the block mask. In particular, Figure 3.4 summarizes design rules listed in [20] to avoid several typical constructs that will cause the above MRC errors. We will follow these design rules and assume no such constructs exist in the layout.

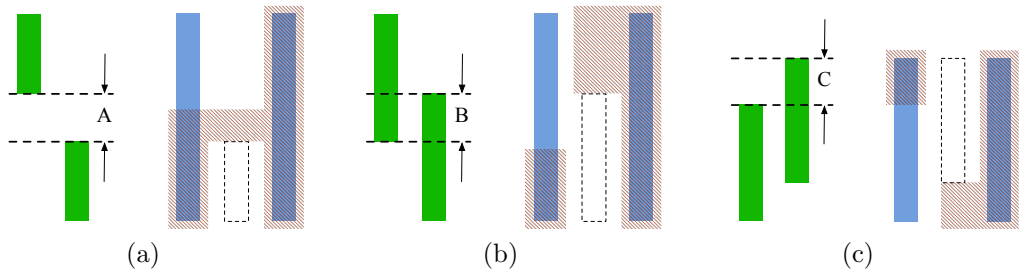


Figure 3.4: Design constructs and corresponding SADP decompositions. Sidewalls are not shown for clarity. (a) Adjacent-track line tip to tip. Rule: $A \geq d$. (b) Parallel run length of adjacent tracks. Rule: $B \geq s$. (c) Step height of non-aligned tips. Rule: $C = 0$ or $C \geq d$.

3.3 SADP Decomposition Algorithm for Row-based Standard Cell Layout

3.3.1 Overview

In this section, we address the problem by first introducing an algorithm for a single standard cell row. The proposed algorithm mainly consists of three stages. In the first stage, the standard cell row is evenly divided into a string of consecutive regions, where each region has the fixed height H of a standard cell, and the same width W. How the value of W is determined will be discussed in Section 3.3.4. In the second stage, decompositions of each region will be computed. In the third stage, we will construct a solution graph according to the region decompositions computed from the previous stage. After the graph is constructed, a shortest path will be found and transformed to a final SADP decomposition.

The motivation of our algorithm is as follows. In the first stage, by dividing the standard cell row into uniform regions, we can upper-bound the number of features inside a region as a function of H and W, which are both constants. Although the SADP decomposition problem is NP-hard in general, by bounding the number of features, we can find all the region decompositions in a fixed amount of time. Moreover, a cost is associated with each decomposition and reflects the total number of overlay violations. In the second stage, the problem is formulated as a single source shortest path problem. It can be shown that the shortest path of the solution graph corresponds to an optimal solution of the decomposition problem, and it can be found efficiently in polynomial time.

In the following sections, we will discuss the details and concepts used in each stage, and show the correctness and complexity of the proposed algorithm. Note that we will discuss the first stage after the second and third stages, since the value of W is a result of the idea described in the latter two stages. Thus, we assume W is available during the introduction of these two stages. To illustrate the overall idea, an example is shown in Figure 3.8–3.11. An analysis of the algorithm is given in Section 3.3.6.

3.3.2 Layout Decomposition for Regions

We start by considering how to pattern a single feature without any overlay violation. There are two available methods. The first method is to simply place a core pattern that has the exact same shape as the feature at its location, and follow the SADP process to pattern the feature. We refer to such an approach as *assigning a core* (pattern). The second method is not as obvious. An *auxiliary core* pattern that has the shape of the bloated contour of the feature is placed around the feature, such that the sidewalls generated can define the feature. Since core pattern is not directly assigned to the feature, we refer to it as *assigning a space* (pattern) to the feature. To distinguish the core patterns in these two approaches, we refer to a core pattern that is assigned in the first approach as a *main core*.

The challenge happens when there are multiple features. The design rule limits the flexibility of assigning cores or spaces to the features. When the distance between two features is smaller than s, we cannot assign cores to them simultaneously. It is tempting to model the problem as a graph two-coloring problem similar to the classic approach to LELE double patterning. However, there is a fundamental difference between them. First, as we have shown in the previous chapter, a graph two-coloring solution is only a necessary condition to the existence of a SADP decomposition. Second, a graph with odd cycle does not have a two-coloring solution, and LELE utilizes a stitching technique to break a feature into several to resolve the odd cycle. In SADP, we utilize the merging technique to break the odd cycle, which may successfully decompose the layout with added overlay violations.

Given the fact that each layout region has a fixed width W and height H, the maximum number of features will be bounded to a constant k. Thus, we can find all decompositions of a region in a fixed amount of time. We have the following cases to consider:

1. For each feature, we assign a core pattern or auxiliary pattern. Note that the potential conflict will be minimized when the width of the auxiliary core pattern is set to the minimum value d.

2. We merge a pair of conflicting features when they are both assigned as cores. In particular, a minimal core pattern is 'padded' between the conflicting features edges such that the two features are connected as

a single pattern. The number of overlay violations caused may be:

0: Both edges are tips.

1: One of the edge is a tip and the other a side.

2: Both of the edges are sides.

An example is given in Figure 3.5(b). We refer to such case as 'core-core-merge' (CCM).

3. Conflict may also exist between a main core and an auxiliary core. There are two cases according to their relative positions:

   - The main core is part of the auxiliary core, and the sides of the main core align with the sides of the auxiliary core. In this case, the conflict can be resolved by merging both cores, as the tips of the main core can be defined by block mask without causing overlay violation. Figure 3.5(c) shows such a case, in which feature B is assigned as core and merges with the auxiliary core of A. We refer to this case as 'core-aux-merge' (CAM).
   - Otherwise, the conflict must be resolved by either:
     (a) Removing the conflict part of the auxiliary core, and post-processing the auxiliary core such that its shape is valid. This is illustrated in Figure 3.5(d). We refer this case as 'core-aux-removal' (CAR).
     (b) Merging the cores along the sides. This is illustrated in Figure 3.5(e). This is a special case of CAM.

     Either case will cause one edge of either feature defined by the block mask. The case when the block-defined edge is a tip is preferred since this will not increase the violations. Otherwise, an overlay violation will be caused.

4. Finally, two auxiliary cores can be safely merged together directly or by core-padding. We refer to this as 'aux-aux-merge' (AAM).

The above cases are illustrated in Figure 3.5. Following the above procedure, we can find all the decompositions. A number of overlay violations will be created in each decomposition.

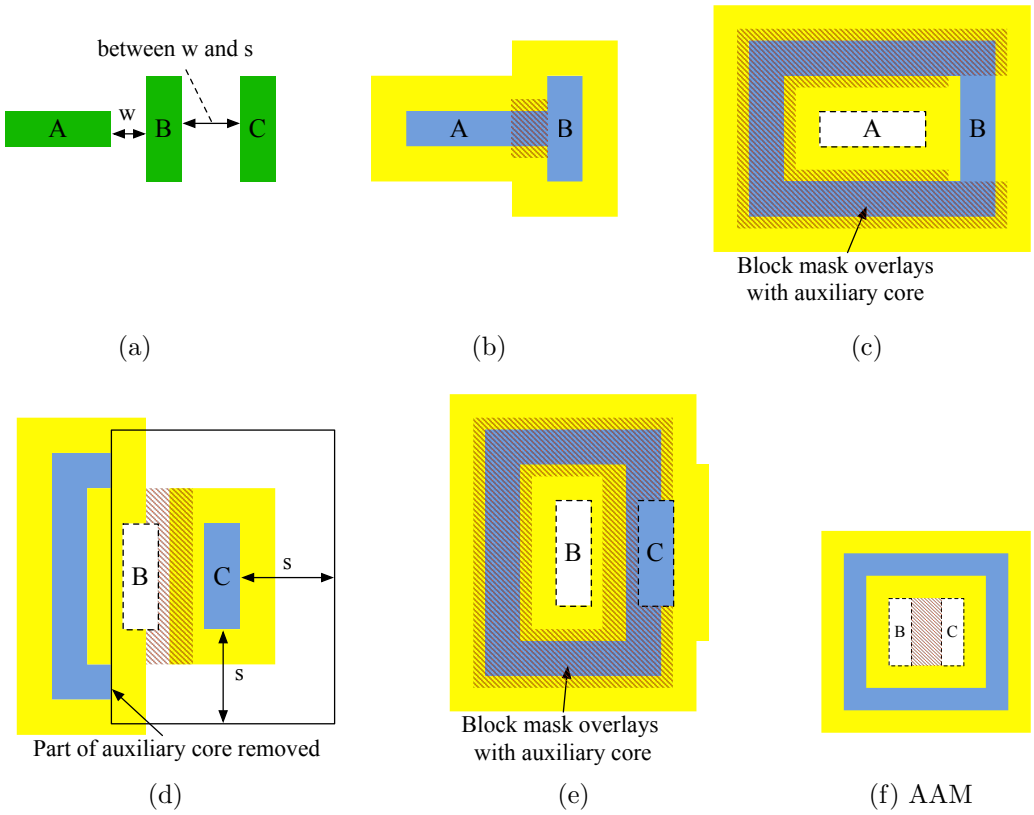


Figure 3.5: (a) Layout to decompose. (b) Merging main cores (CCM). One violation is caused at feature B's left side. (c) Merging assigned and auxiliary cores (CAM). No violation is caused since the block defines the tips. (d) Removal of the conflicting part of the auxiliary core (CAR). The outer box around C denotes the region of minimum distance s. (e) Merging conflicting main core and auxiliary cores (CAM). One violation is caused at feature C' s left side. (f) Merging two auxiliary cores (AAM).

Note that as we computed the decompositions for each region independently, parallel processing can be easily adopted to speed up the whole process. Thus, the potential parallel scalability of the algorithm is huge.

3.3.3 Solution Graph

After the previous stage, the decompositions for each region are found. We follow by constructing a *solution graph* as follows. A solution graph is defined as a directed acyclic graph, where each vertex corresponds to a decomposition, and each arc between two vertices denotes a pair of compatible decompositions. Both of the vertices and arcs have costs associated. The cost of a vertex u is defined as the number of overlay violations in the corresponding decomposition. Let $D(u)$ be the decomposition w.r.t. vertex u, and $R(u)$ is corresponding region. An arc (u, v) will be added between vertices u and v if and only if $D(u)$ is left adjacent to $D(v)$ and u is *compatible* to v. Two decompositions are said to be compatible if they can be combined as a decomposition for both of the regions. We denote $D(\{u, v\})$ as the combination of $D(u)$ and $D(v)$. The arc cost $c(u, v)$ is defined as the number of overlay violations introduced when combining the decompositions. We add a dummy source vertex u_0 with cost 0, and a set of arcs (u_0, v) with cost 0 for all v where $R(v)$ is the left-most region in the standard cell row. Similarly, we add a dummy sink vertex v_0 with cost 0, and a set of arcs (u, v_0) with cost 0 for all u where $R(u)$ is the right-most region in the standard cell row.

Note that in the above, the width of the regions is chosen in such a way that a region decomposition will not affect decompositions from a non-adjacent region. Therefore, arcs will only be added between the vertices from two adjacent regions.

The most critical issue we need to handle when constructing the graph is to find the arcs and compute their weights. This involves finding the compatibility two decompositions and resolving the potential conflict when combining. The algorithm of combining decompositions is in fact the same as the algorithm we introduced in Section 3.3.2. In particular, we only need to examine the features near the cut line. Again, there are two situations to consider: merging the core patterns and removing the conflicting core patterns. The overlay violations introduced during the combination process

will be the costs of the arcs.

After the graph is constructed, we find a shortest path from u_0 to v_0 in the solution graph. We have the following theorem:

Theorem 4. *A shortest path in the solution graph corresponds to a complete SADP decomposition of the layout with minimum overlay violations.*

Proof. We first show that a path from u_0 to v_0 corresponds to a complete SADP decomposition. This can be proven by showing such a path contains exactly one vertex from each region, and all the corresponding decompositions are compatible.

In the solution graph, the arcs only connect vertices from adjacent regions. Thus, a path from u_0 to v_0 must have at least one vertex from each region. Given any vertex v except u_0 and v_0, all of its incoming arcs must be from some vertex u where $R(u)$ is left-adjacent to $R(v)$ (we can view $R(u_0)$ as an empty region left-adjacent to the left-most region). Similarly, all of the outgoing arcs of v must enter some vertex w where $R(v)$ is left-adjacent to $R(w)$ (we can view $R(v_0)$ as an empty region right-adjacent to the right-most region). Thus, if v is in the path, no other vertex v' in the same region (where $R(v) = R(v')$ and $v \neq v'$) will be in the path, for otherwise there will be a path from v (v') to the predecessor of v' (v), but the arcs can only between two adjacent regions. In conclusion, exactly one vertex from each region will be included in the path. Thus, a path defines a complete decomposition and vice versa.

Since we have all possible decompositions for each region, a complete decomposition must be a combination of the decompositions chosen from each region. If there is decomposition with smaller cost, we can then transform it back to a path with smaller cost. Thus, the shortest path corresponds to a complete decomposition with minimum overlay violations. □

3.3.4 Standard Cell Row Partitioning

In the first stage of the algorithm, the standard cell row will be divided into multiple manageable regions such that SADP decompositions can be found efficiently. The motivation is two-fold. First, we would like to find all decompositions of a region efficiently. Assume the height of the standard cell is H, and the width of each region is no more than W. Due to the minimum

width rule, the number of features inside a region must be bounded. We can thus find all the decompositions within a region within a limited bound.

The second point is critical to the following stages of the algorithm. As we introduced in Section 3.3.1, we rely on a solution graph to find a global optimal solution, which corresponds to a shortest path of the graph. Each vertex in the shortest path represents a decomposition of a region, and each region must have exactly one decomposition (vertex) in the path. Recall that the arcs in the solution graph represent the weight of combining two vertices and thus the combined decompositions of two regions. If two regions are too close, we have to determine whether an arc must be added between them since there may be conflicts between some features in the regions. Thus, we need to set W to be large enough such that the non-adjacent regions will never conflict with each other. In such a way, arcs will only be added between adjacent regions. Visually, if we 'list' all the decompositions (vertices) of a region vertically, and each of these lists horizontally from left to right, we can clearly see that arcs only exist between adjacent lists of vertices. A path from the dummy source to the dummy sink nodes will visit exactly one vertex in each list once. To determine the appropriate value of W, there are two cases to consider.

Direct Case. We first discuss the case where two non-adjacent regions *directly* affect each other. Consider three adjacent regions R_1, R_2 and R_3, where R_1 is left-adjacent to R_2 and R_2 is left-adjacent to R_3. In the worst case, there will be a feature A aligned at the right cut line of R_1, and a feature B aligned at the left cut line of R_3. In order to avoid interaction between R_1 and R_3, there are three cases to be considered.

1. Features A and B are both assigned to be cores. Clearly, when $W \geq s$, A and B can be patterned independently without conflict. This is illustrated in Figure 3.6(a).

2. One of the features is assigned as core and the other is as space. Without loss of generality, assume A is core and B is space as in Figure 3.6(b). In this case, the decomposition for R_3 needs to use an auxiliary core pattern for the left edge of B. When $W \geq s + d + w$, the auxiliary core will not affect A.

3. Both of the features are assigned as spaces in their decompositions. In

this case, an auxiliary core will be generated for A and B, respectively. When $W \geq s + 2d + 2w$, the auxiliary cores will not affect each other, and thus the two regions will not interact.

Thus, we have to set W to be at least $s + 2d + 2w$ to avoid a *direct* interaction between two regions that have one region in between.

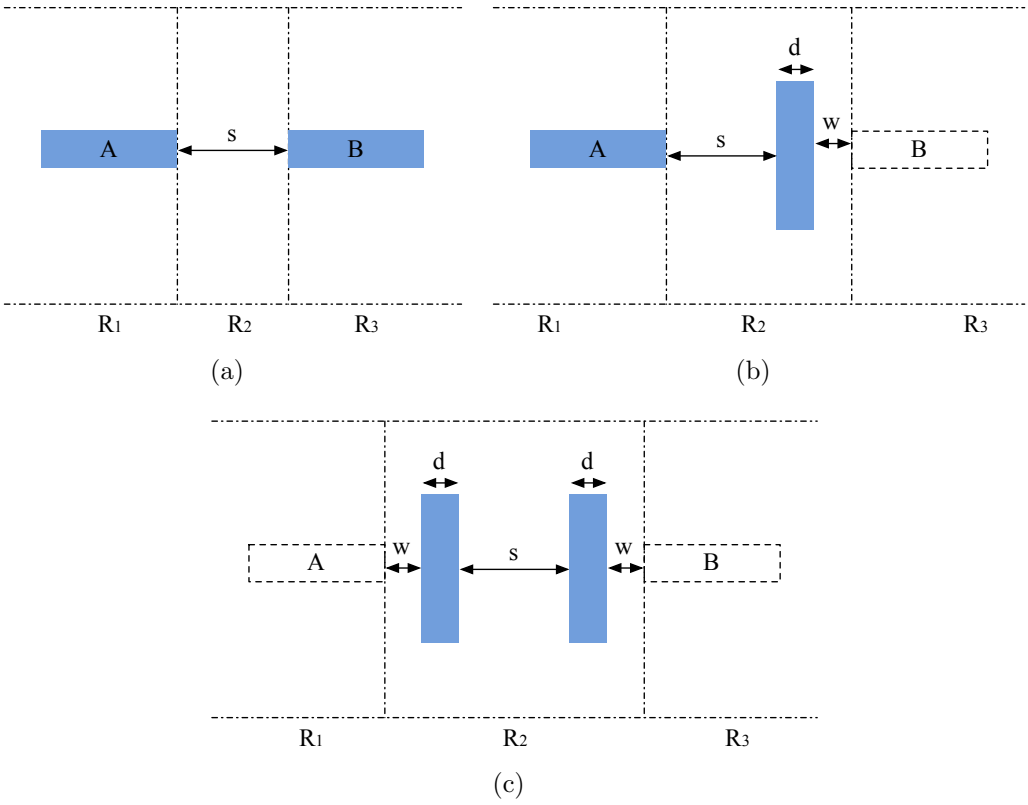


Figure 3.6: Direct cases to determine the value of W. (a) Core-core case. (b) Core-space case. (c) Space-space case.

Indirect Case. The second case considers indirect interaction between non-adjacent regions. Again, consider the three regions in the previous discussion. It is possible that the decomposition combination of R_1 and R_2 may affect the decomposition combination of R_2 and R_3, and we consider this as *indirect*. However, as the following theorem shows, when the W is large enough, the combinations is 'local' enough.

Theorem 5. *Given a region, when its width W is at least $3s + 2d + 2w$, the decomposition combination of it and its left-adjacent region will not affect the decomposition combination of it and its right-adjacent region.*

Proof. The possible interaction between the two regions includes merging cores and removing conflicting cores. Both cases modify the original decomposition by either adding or removing core patterns and adding block patterns.

In the case when cores are being merged, it can happen in core-core, core-aux or aux-aux pair. Consider the regions R_1 and R_2 in the previous section, where R_2 is further divided into three sub-regions R_2^l, R_2^c and R_2^r by lines l and r. The three regions have widths $s+w+d$, s and $s+w+d$, respectively (Figure 3.7). To reach the maximum possible place R_1 can affect in R_2, there must be a feature assigned as space at the right boundary of R_1. In this case, an auxiliary core may be created to help generate the sidewall that can define the right side of feature A. If there is a core pattern to be merged with the auxiliary core, R_2^l must contain a part of it, otherwise they are separated and can be assigned as core simultaneously. Thus, an extra core pattern will be padded, and it is completely *contained* in sub-region R_2^l. As a result, a block pattern must also be added, which will again be completely *contained* in sub-region R_2^l. Since the region R_2^c has a width s, the extra core and block pattern will not affect anything in R_2^r. In other words, any new overlay violations only appear in R_2^l and will not affect R_2^r. Symmetric case holds for R_2 and R_3 where the interaction between R_3 and R_2 only exists in R_2^r and R_2^l is not affected.

In the case when an auxiliary core is being removed in R_2, it happens when there is an main core in R_1. This is in fact a weaker case than the above, as the main core can only affect up to a distance s in R_2^l, while the width of R_2^l is $s+d+w$. The symmetric case holds for R_3 and R_2.

Note that in the above, the width requirement is tight and the total width of three sub-regions is $3s+2d+2w$. □

As a result of Theorem 3.3.4, we have to set W to be at least $3s+2d+2w$ to avoid the *indirect* interaction between two non-adjacent regions. Combining both direct and indirect cases, we will use $3s+2d+2w$ as the least value of W.

Due to the uniform width of the regions, some features may be cut into two (or more) regions. When constructing a solution graph, we need to take special care of such features and make sure to assign consistent core or space to these features.

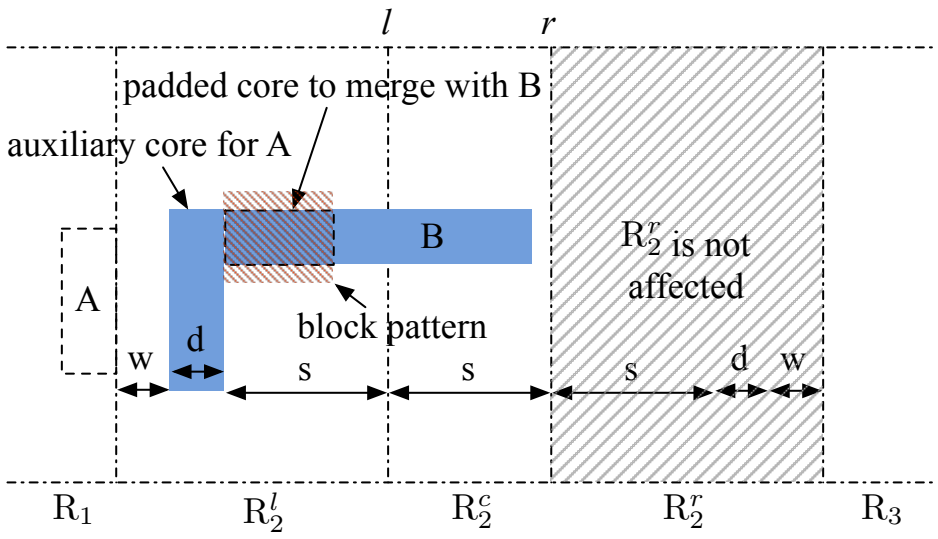


Figure 3.7: Indirect case when cores are merged.

3.3.5 Illustration of the Proposed Algorithm

Figure 3.8–3.11 show a complete example to illustrate our algorithm. For the layout in Figure 3.8, it is divided into three regions R_1, R_2 and R_3. Note that feature A is cut into two pieces A_1 in R_1 and A_2 in R_2.

The decompositions of R_2 are shown in Figure 3.9, as well as the vertices and the costs. In the vertices, 'C' denotes a core and 'S' denotes a space. For example, the vertex in Figure 3.9(c) shows 'SC' and '1'. It means A_2 is assigned a space and B is assigned a core, and the cost (overlay violation) is 1. Also, the block-defined-edges are marked with a red solid line. This clearly shows the overlay violations in each decomposition. The decompositions of R_1 and R_3 are omitted for their simplicity.

In Figure 3.10, we combine the decompositions between R_2 and R_3. The overlay violations due to the combination are also marked. Only four combinations are shown for illustration purpose. 'CCM', 'CAM', 'CAR' and 'AAM' refer to the type of combination we discussed before, and the number refers to the violation(s).

Figure 3.11(a) shows the solution graph constructed using our algorithm. We can see that vertex C in group R_1 is only connected to two vertices in R_2 since we need to maintain the consistency of feature A. Similarly for vertex S. All the vertex costs and arc costs are labeled. We can then easily find a shortest path $u_0 \rightarrow C \rightarrow CS \rightarrow S' \rightarrow v_0$. The total cost is 1, which means

we find a solution with only one overlay violation. The corresponding final decomposition is shown in Figure 3.11(b).

3.3.6 Complexity

The complexity of our algorithm is summarized as follows.

Theorem 6. *The proposed SADP decomposition algorithm runs in polynomial time.*

Proof. Since W and H are constants, combined with the minimum width rule, the maximum number of features in a region is limited. In the worst case, there will be at most $k = (WH)/(d+s)^2$ features. Thus, the time needed to find all decompositions of a region is bounded in $N = 2^k$, which is a constant. For two adjacent regions, there will be at most N^2 edges. Finally, finding a shortest path in a DAG can be done efficiently in $O(n)$, where $n = NL/W$ is the number of vertices in the solution graph, L is the length of the standard cell row. In conclusion, the algorithm runs in polynomial-time. □

3.3.7 Decomposition for Multiple-row Layout

In the previous sections, we have discussed the algorithm of solving a single standard cell row. In a full row-based design, the power tracks are always parallel and in alternating order. Naively, we can apply the algorithm to solve for each row, and find a decomposition with consistent core and space assignment to the power tracks that minimizes the violations, which requires enumerating all combinations of row decompositions.

A better approach will be the following. For each row, we compute all optimal solutions. There will be four possible core and space assignments for power tracks, where VDD and GND are assigned as core-core, core-space, space-core and space-space. We again construct a higher-level solution graph. For each row, there will be four vertices with costs. The arcs will be added in a similar way, *i.e.*, the vertices in a row are connected to the vertices in the adjacent row(s). Note that the core and space assignments must be consistent, and there is no cost associated with the arcs. A dummy source

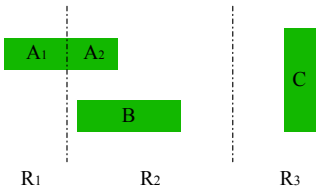


Figure 3.8: An example layout.

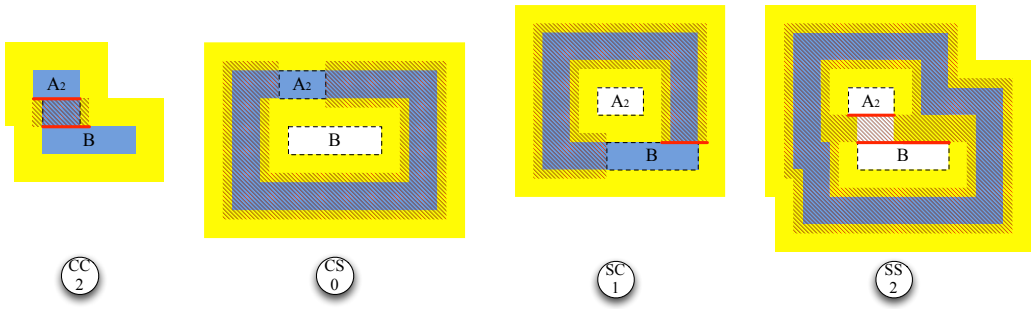


Figure 3.9: Decompositions of R_2.

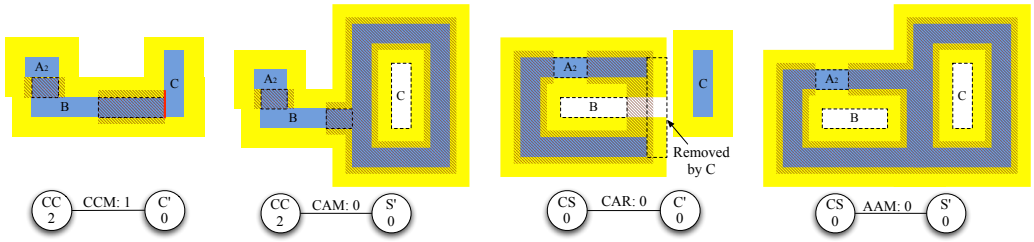
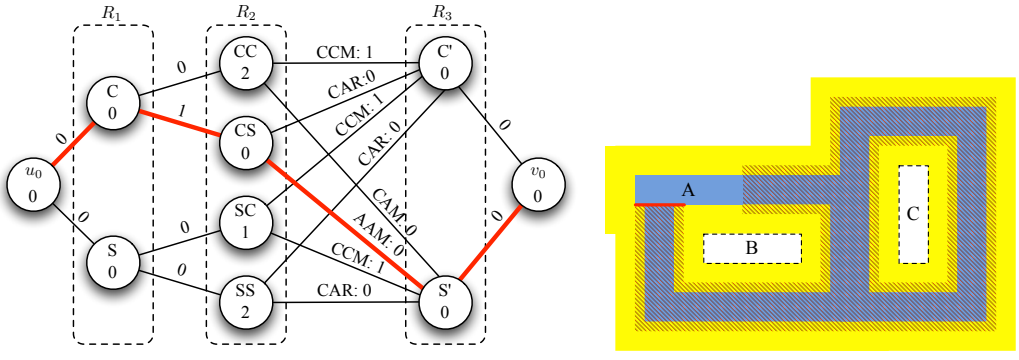


Figure 3.10: Combinations between decompositions in R_2 and R_3. Only part of the combinations are shown.



(a) Solution graph and the shortest path. (b) Corresponding decomposition.

Figure 3.11: Final solution.

and sink vertices will also be added in a similar fashion. Clearly, the shortest path from source to sink is the full decomposition.

By using the above algorithm, a global optimal decomposition can be found for a multiple-row layout. The time complexity remains polynomial.

3.4 Experiments

The proposed algorithm is implemented in C++ and run on a Linux machine with 2.8 GHz CPU and 8 GB RAM. We used Nangate Open Cell Library [32] as our starting point to create the benchmarks. The standard cells are first scaled to reflect 14 nm technology node, where we set $w = d = 20$ nm and $s = 50$ nm. The relative feature locations are adjusted to adhere to the SADP design rules and SADP-compliant design style as discussed in Section 3.2.3–3.2.4. A set of benchmarks with different row lengths are created by randomly placing the cells consecutively in a row. Connections between cells are randomly added between the I/O pins.

Table 3.1 shows the experimental results. The second and third columns show some statistics of the benchmark. The fourth and fifth columns show the results of running our algorithm. From the run time we can see that it is nearly linearly correlated with the number of cells in the benchmark, which demonstrates our algorithm scales well. Our algorithm can solve the medium size benchmark (10k cells) within 5 minutes. Note that as the library is not optimized for SADP, some violations are expected. However, the average violations per cell remains reasonably small.

Table 3.1: SADP Layout Decomposition Results

Name	#Cells	#Features	#Violations	Runtime (s)
tiny	100	296	114	4.88
small	1000	2984	1025	69.52
medium	10000	30018	12890	500.76
large	100000	300634	109424	6123.4

3.5 Conclusion

In this chapter, we discussed the SADP decomposition problem for row-based standard cell layout and presented our solution. In contrast to previous works, which tried to minimize or disallow overlay, we aim at minimizing the overlay violation. A polynomial-time optimal algorithm is proposed to solve the problem by utilizing the characteristics of standard cell design. The major techniques in our algorithm include layout partitioning, which breaks down the overall problem complexity, and a solution graph formulation, which captures the region-to-region interaction and can be used to find an optimal solution from shortest paths. The experimental results showed that our method can solve large scale problems in a relatively short time. Moreover, our algorithm can be easily parallelized. As row-based standard cell design is a major choice in ASIC design, our approach is expected to find its potential in the manufacturing industry for the sub-20 nm technology node.

CHAPTER 4

DSA DESIGN-TECHNOLOGY CO-OPTIMIZATION

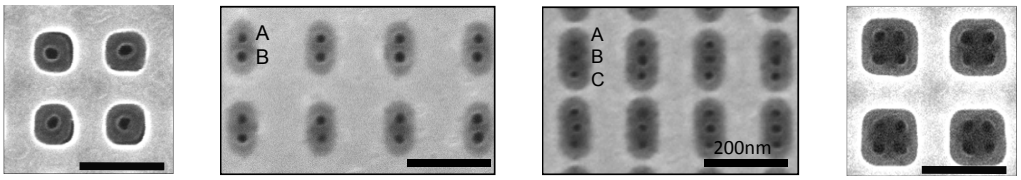
4.1 Introduction

4.1.1 Background

With the continuous scaling down of semiconductor technology, lithography has become the bottleneck for integrated circuit (IC) fabrication. To make IC designs manufacturable for the sub-14 nm technology nodes, the semiconductor industry has to adopt advanced lithography technologies, such as extreme ultraviolet lithography (EUV), electron beam lithography (EBL), multiple patterning lithography (MPL) and block copolymer directed self-assembly (DSA). EUV has been proposed and studied yet still is not mature for industrial-scale implementation. EBL has relatively small throughput and thus is only suitable for small volume production. Multiple patterning lithographies, such as double patterning [22], [26], [29], triple patterning [35] and quadruple patterning [36], are the current industry standards. However, DPL ultimately reaches its limit and as more masks are involved, the manufacturing cost may become prohibitively high.

DSA is considered a very promising technology for patterning contact holes and vias in 7 nm technology nodes [6]–[8], [37]. In DSA process, the contact holes and vias are formed by the annealing process [10], [12] guided by the "guiding templates". The guiding templates are patterned with traditional optical lithography process such as 193 nm immersion lithography (193i), which has a coarser pitch resolution. The guiding templates play the role of controlling the DSA patterns formed, which will have a finer resolution than the templates. The DSA contact pitch depends on the chemical property of block copolymer and it can be adjusted within a certain range under strong lateral confinement to deviate from the natural pitch. As a result, different

patterns can be obtained through various parameters. However, the variation in the template shapes will easily affect the locations of the final contact holes. In particular, small variations at critical boundaries of the guiding templates may result in huge interference on the final DSA pattern. We refer to the templates as *infeasible* when they have variation larger than some threshold value and thus cannot print the DSA patterns reliably. For those templates that can print DSA patterns with tolerable variation, we refer to them as *feasible templates*. Figure 4.1 shows several templates that are successfully applied in patterning for contact hole fabrication [6].



(a) Single hole (b) 2-hole (c) 3-hole (d) 4-hole

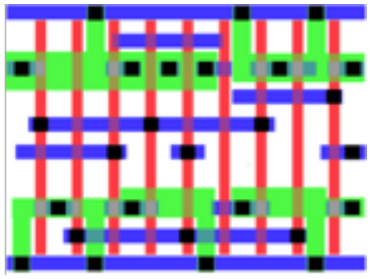
Figure 4.1: SEM picture of different guiding templates with their DSA pattern [6]

Since the DSA technology is very sensitive to the shapes and distributions of patterns, it is necessary for the EDA engines to understand the DSA process such that layouts can be optimized to be DSA-compliant. This motivates us to consider design-technology co-optimization (DTCO) for DSA. In this chapter, we explore the contact layer and cut layer optimization problems for DSA, and propose methods to solve them.

4.1.2 Contact Layer Optimization for DSA

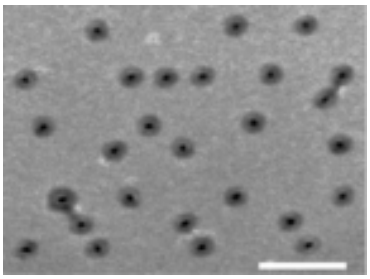
The industry has been transitioning from random 2D designs to highly regular 1D gridded designs for sub-20 nm nodes for its larger process window and higher yield [38]. In these designs, the challenge mainly lies in following the design rules in the poly/metal1 layers. To connect these layers, usually Metal1 (M1) layer is used for horizontal connections and local interconnect (LI) layer is used for vertical connections. Whenever a direction switch happens between M1 and LI, a contact is inserted. As a result, the contact layer can be highly dense and random in these designs, which is beyond the capability of traditional lithography. On the other hand, it has been shown

that DSA can be used for contact hole patterning [7]. Figure 4.2 shows an example of using DSA to pattern the contacts of a half-adder. At the 22 nm node (Figure 4.2(c)), each contact can be patterned by a single-hole template. When the pitch is at the 7 nm node (Figure 4.2(d)), a large template can be used to pattern multiple close contacts. Clearly, it is desirable to utilize DSA to pattern the dense contact layer.

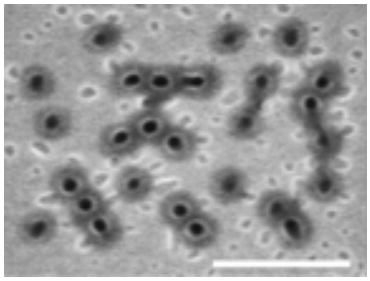


(a) Half-adder layout

(b) Contact layer



(c) Patterning at 22 nm



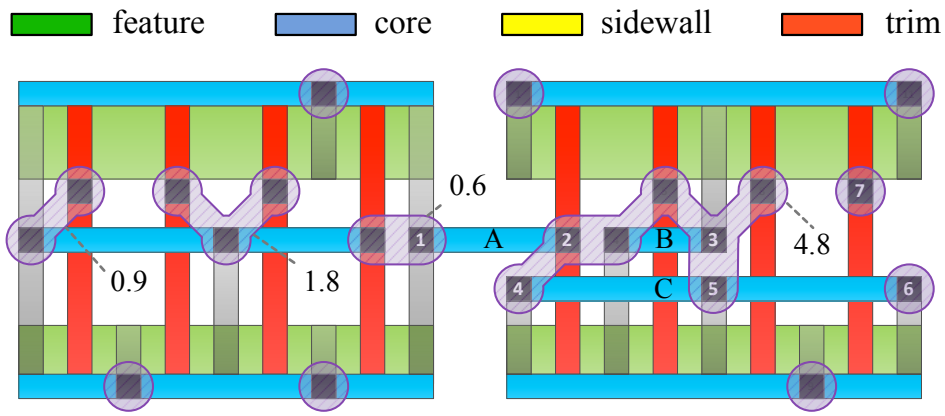
(d) Patterning at 7 nm

Figure 4.2: DSA contact patterning at 22 nm and 7 nm technology node and quality of DSA pattern formation. Scale bar: 200 nm.

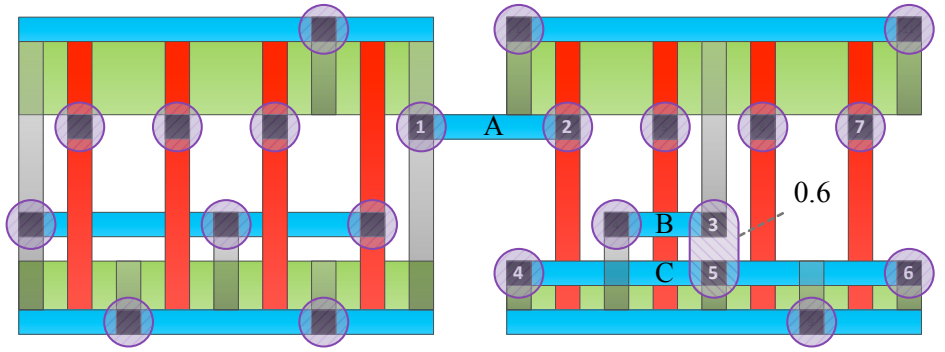
To pattern the contact layer with DSA, only the feasible guiding templates should be used. Du *et al.* proposed a contact layer optimization algorithm for 1D standard cell library [9]. However, their work only focused on cell level optimization. Even with an optimized standard cell library, infeasible templates may be introduced after the physical design phase. For example, routing may create new contacts when switching from LI to M1, which may introduce new infeasible templates.

In this chapter, we discuss the contact layer optimization problem for DSA in full chip level layout. We propose a cost function that models the variation of guiding templates. We further identify the constraints in the problem and propose an efficient optimization algorithm based on simulated annealing. The experimental results show that our algorithm is able to optimize the

contact layer for DSA manufacturing effectively. To the best of our knowledge, there is no previous work on contact layer optimization targeting on full-chip level. An example of layout optimized by our algorithm is shown in Figure 4.3.



(a) Before permutation



(b) After permutation

Figure 4.3: The layout before and after wire permutation.

4.1.3 Cut Layer Optimization for DSA

The advantage of DSA is not limited to contact hole patterning. In particular, the use of DSA can be further expanded for cut printing in 1D design. Due to the high regularity of the dense lines, a set of optimal lithography conditions can be well-defined to print the dense lines with high image quality by a variety of lithography techniques, such as self-aligned double-patterning (SADP). A cut layer consists of a number of identical cut patterns, each located at the line-end of a target wire. An example of such a design is illustrated in Figure 4.4.

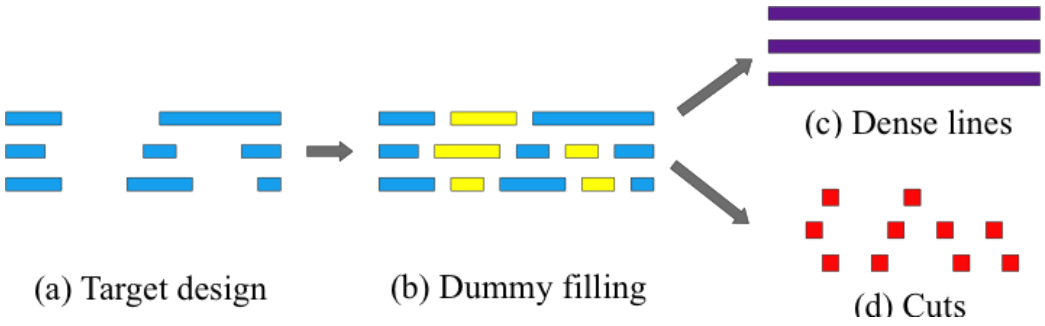


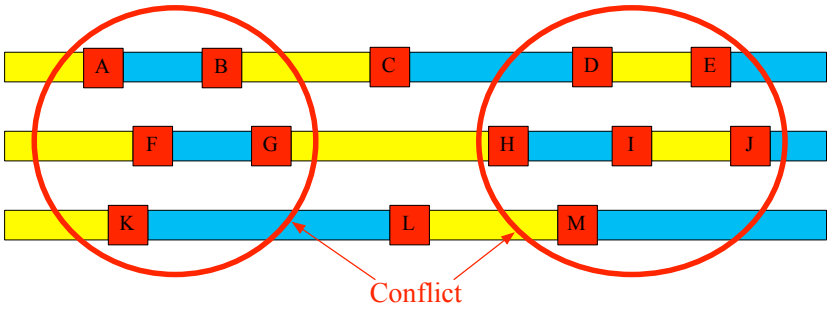
Figure 4.4: 1D design fabricated by a combination of dense lines and cuts.

The randomness of logic circuits will affect the cut pattern distribution and introduce a major challenge in fabricating 1D gridded designs. Although the guiding template shapes can be arbitrary, the overlay accuracies of the contact holes are different and largely depend on the templates. To utilize DSA for cut printing, cuts that have pitch smaller than the lithography pitch should be grouped and patterned in a guiding template. However, a design that is not DSA-compliant may introduce infeasible templates. We observed that for a 1D design, the line-ends of each target wire are usually allowed to be extended within a certain range. Such extension does not impact logic connections. We refer to this technique as line-end extension. Through line-end extension, cuts can be redistributed such that we can pattern the cut layer using only feasible templates.

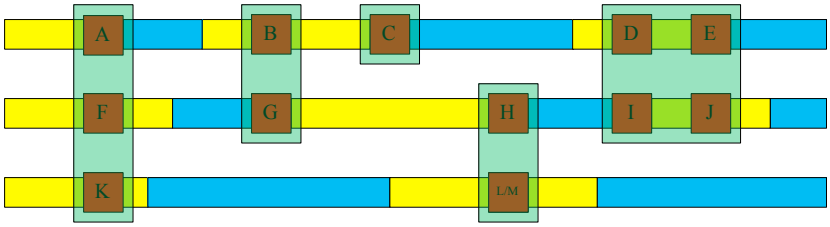
Figure 4.5 shows an example of cut redistribution. The original cuts are shown in Figure 4.5(a), where some of them have conflicts and violate the design rules. The real wires are colored in blue while the dummy wires are colored in yellow. Figure 4.5(b) shows a cut redistribution solution, where the possible templates are as shown in Figure 4.1. To illustrate that the real wires are 'extended', we keep the wires' colors to show the difference. Note that the cuts L and M are *merged* together, which is allowed as long as the metal wires are not shorted.

We refer to the above problem as *DSA cut redistribution problem*. We propose an efficient algorithm to optimize cut layers for DSA patterning. We utilize the line-end extension technique to perturb the original cuts such that they can be redistributed and grouped into valid DSA templates, and the infeasible templates are eliminated. To the best of our knowledge, this is the first work to address the DSA cut redistribution problem.

The rest of the chapter is organized as follows. Section 4.2–Section 4.3



(a) Original cuts.



(b) Cuts after DSA redistribution.

Figure 4.5: An example of cut redistribution.

present our work on contact layer optimization. We first present a cost modeling and some design constraints in Section 4.2, then introduce the details of our simulated-annealing based optimization scheme in Section 4.3. We discuss the DSA cut redistribution problem in Section 4.4–Section 4.5. Specifically, Section 4.4 gives a formulation of the problem. Section 4.5 presents the details of our algorithm. Section 4.6 reports the experimental results. Finally, conclusions are drawn in Section 4.7. The work in this chapter is published in [39] and [40].

4.2 Cost Modeling and Design Constraints

4.2.1 Cost of DSA Template

To pattern the contact holes using DSA, guiding templates should be printed first with conventional lithography such as 193i that has a coarser pitch. Contact holes are then patterned by DSA process. The controlling parameters will determine the hole pitch within the guiding template. Clearly, when the contact pitch in the layout is large enough, each contact can be defined by

a guiding template separately as shown in Figure 4.2(c). However, when the contact pitch gets smaller, some of the contacts must be grouped together and patterned in a guiding template as shown in Figure 4.2(d). There are several challenges that may affect the quality of the DSA patterned:

1. The lithography variation may introduce variation in template shapes and ultimately affect the pitch of the patterned holes. In the worst case, small variations at critical boundaries of guiding templates may result in drastic differences in the final DSA pattern generation. Usually, the more holes in a template, the harder it is to control the variation of the patterned holes. For example, two-hole template has larger variation than single-hole template.

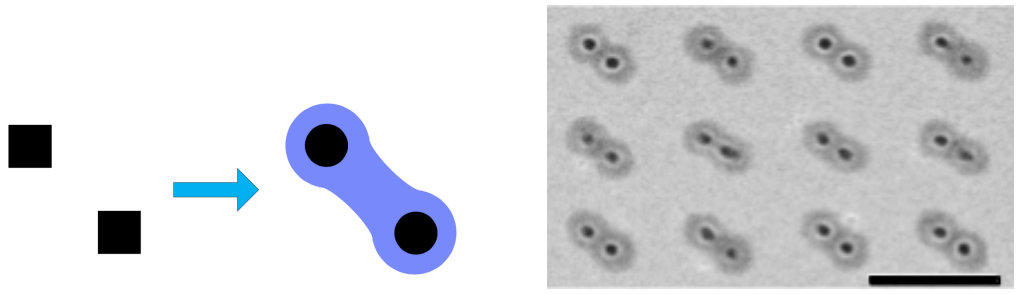
2. Some contacts may form diagonal patterns, which are difficult to print as they have irregular pitch while the contacts that are on the same row (column) have regular pitch. Regular pitched contacts can be defined easily by a simple rectangular shaped template, *e.g.*, Figure 4.1(b). On the other hand, if the contacts are in diagonal position, their pitch will be larger than the maximum pitch of DSA holes. Thus, they need to be put into two separate single hole templates that are very close to each other, which cannot be printed by conventional lithography. To pattern this diagonal pair, we need to use a special peanut-shaped template as shown in Figure 4.6 [41]. Such a template shape is difficult to print by the conventional lithography and suffers from a large variation.

3. In a real layout, the density of the contacts will also affect the formation of the contacts and introduce variation.

To model the variation and patterning difficulty of a DSA template i, we can define a *template cost* c_i. Clearly, challenge 1 indicates a larger template will have a larger cost; challenge 2 shows that a more irregular template shape should have a larger cost. We define c_i as follows:

$$c_i = w_s \times s_i + w_p \times p_i + w_l \times l_i + f(i) \tag{4.1}$$

where s_i denotes the template size, p_i denotes the number of diagonal-shaped pairs in the template, l_i refers to the local density around template i, which is defined as the number of contacts over the area of a fixed window centered



(a) (b)

Figure 4.6: (a) A diagonal pair of contacts much be guided by a 'peanut-shaped' template. (b) Peanut-shaped templates may lead to large overlay accuracy variations. Scale bar: 200 nm.

at the contact. $f(i)$ is a penalty function to disallow infeasible templates. In practice, $f(i)$ returns a prohibitively large number if i is infeasible, or a small number (or zero) otherwise. w_s, w_p and w_l denote the weight of size, number of diagonal shapes and local density terms, respectively. In this work, we define s_i as the number of holes in the template minus 1, since when the local density is 0, a single hole template will have zero cost, which is consistent. Finally, the cost of a layout is defined as the sum of the total template costs.

4.2.2 1D Design Optimization via Wire Permutation

In 1D standard cell design, M1 wires with contact can be permuted row-wise as long as the logic is the same as the original circuit [9]. In a full-chip layout, the intra-cell connections should be considered. For example, wire A in Figure 4.3(a) is an inter-cell connection, which is not considered in [9]. As a result, contact 1 can be distributed on any track in their work. However, this is not valid since contact 2 can only utilize the free tracks between P-active and N-active, which limits the candidate locations of contact 1. We summarize the constraints of wire permutation as follows:

1. Geometry constraint. The permuted M1 wire cannot overlap with other M1 wires.

2. Contact constraint. The contacts that are used to connect the polysilicon cannot be placed on the gate regions. This implies the underlying

metal wires can only utilize the routing tracks between N-active and P-active. As illustrated in Figure 4.3(a), contact 2 is on wire A and above polysilicon. Thus, wire A can only be permuted inside the free track region.

3. Vertical constraint. This is the same as the vertical constraint in the classical channel routing problem. If there are two contacts in the same column, there will be a vertical constraint between them. This implies that one metal wire must be above the other. For example, contact 3 is above contact 5 in Figure 4.3(a). Therefore, wire B must be above wire C.

4. Track constraint. Intra-cell connections are used for PMOS/NMOS connections. Thus, a wire that is connecting PMOS (NMOS) can only utilize the tracks that lie in the P-active (N-active) and the free tracks between N-well and P-well. A wire that is used for inter-cell connections can utilize any routing tracks. For example, wire C is used for NMOS connection, and thus it can only be permuted within the free track region and the N-active region.

Clearly, the feasible move range of a wire is the intersection result from the above constraints.

The costs of the non-single hole templates are labeled in the figures (single hole template has cost 0). In this work, we set $w_s = 0.6$, $w_p = 0.3$, $w_l = 0.1$. The local window size for density computation is 5. Note that for clarity and illustration purpose, we ignore the local density in the example used in this chapter. We can see that the original cost of the layout before permutation is 8.1, while the cost of the modified layout after permutation is 0.6. Note that the very large and irregular template in the right cell could have an infinitely large cost by specifying $f(\cdot)$.

Another important consideration is the routing from M2 and above. Clearly, M1 in post-routing layout may have connections to M2. Permuting M1 may affect the original logic. Thus, we need to take special care about inter-cell routings. Fortunately, usually most of the inter-cell routings switch layers early and will only utilize a small fraction of M1. In practice, we can either fix these M1 wires, or obtain a feasible move range by considering constraints from M2 and above.

4.2.3 Problem Description

We now define the DSA contact layer optimization problem in full-chip layout as follows.

Definition 3. *Given a contact layer in 1D design, optimize the layer via wire permutation such that it can be patterned by the DSA process, and the total template cost is minimized.*

4.3 DSA Contact Layer Optimization for Full-chip Layout

It is shown that the DSA contact layer optimization problem is NP- hard [9]. The previous SAT formulation can solve the problem in cell scale, but not full-chip level. On the other hand, simulated annealing [42] has been successfully employed in many area, such as floorplanning problem [43]. In this work, we propose a simulated-annealing based algorithm to search for a near-optimal solution iteratively. We introduce our annealing scheme and the operations (moves) to search for neighbors of a state (solution) in the following. As in any simulated annealing based algorithm, the efficiencies of evaluating cost (energy) and updating the solution directly affect the efficiency of the algorithm as they are evaluated at each iteration during the cooling. Therefore, we propose an efficient algorithm to dynamically update the graph connected components that correspond to the template shapes. This enables fast evaluation of template cost.

4.3.1 Overall Annealing Scheme

The proposed algorithm follows the three phases below to converge to a solution:

1. Perturb the given layout and generate candidate solutions.
2. Evaluate the new cost of the candidate solutions.
3. Accept or reject the candidate solutions according to the Metropolis criteria.

4.3.2 Candidate Solution Generation

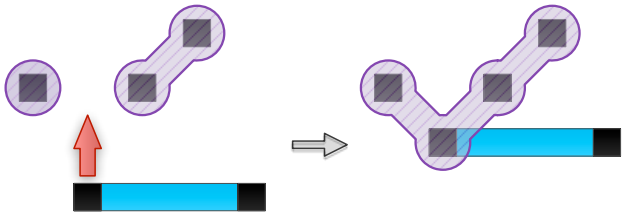
A naive candidate generation operation is to randomly select a metal wire and move it to a feasible location. However, this approach is inefficient due to the same costs of some neighboring solutions. For example, the solution in Figure 4.3(b) will have the same cost as the solution where contact 7 is moved down by one grid. Clearly, there will be lots of such cases. This inevitably increases the annealing iterations.

To avoid oscillating among a set of neighboring solutions with the same costs, guidance is required during the solution generation. We propose a field-based method that is similar to the density penalty concept for overlap minimization in analytical placement. Particularly, we define the local contact density of a grid as the ratio of contacts over the area of a window centered at the grid. The region that has higher densities will be chosen and a local search is applied to search for a minimum cost solution. When the temperature is high, we allow regions with small densities to be chosen. The purpose is to let the algorithm probe at different places in the solution space in higher temperature to avoid being trapped in local minima. As the temperature gets lower, we gradually raise the threshold and only target high density regions. This is assuming that we have reached a ridge in the solution space, at this stage we have explored many sub-optimal regions, and the probability of getting a closer-to-optimal solution is high enough. The region size of the local search does not have to be fixed. Instead, it is adaptively increased when no better neighbor solution can be found.

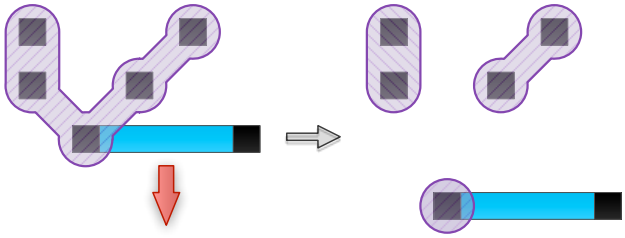
4.3.3 Solution Updating and Cost Evaluating

When wires are permuted, it will be very inefficient if the cost of the whole layout is re-evaluated from scratch since only a small fraction of the layout is locally changed. We propose an incremental cost updating algorithm based on disjoint set data structure [44]. We also keep track of the current size and number of diagonal shapes in a template (component). Suppose the cost of the original layout is known and the connected component is already constructed. If we permute a wire, we have the following cases:

1. A contact on the wire conflicts with some other contacts and thus forms new template shapes. In this case, we can simply set the parent of the



(a) New template shape



(b) Split template shapes

Figure 4.7: Incremental update of cost and template.

existing template shape as the contact on the wire.[1] If there are multiple templates conflicts, they are merged as a single template. The cost can be updated in constant time. This is illustrated in Figure 4.7(a), the left contact on the wire being permuted conflicts with contacts from two separate templates. The two templates and the contact form a new template, which now has four templates and two new diagonal shapes. Thus, the delta cost is computed as

$$\begin{aligned} \Delta cost = (3-1)w_s \quad & (\Delta size) \\ + 2w_p \quad & (\Delta diagonal) \end{aligned}$$

2. A contact on the wire belongs to some template. Permuting the wire splits the template into several new templates. In this case, we need to compute these new templates by running breadth-first-search (BFS), starting from each contact that it is connecting to. Even though the worst case runtime is still linear to the size of contacts, in practice, each template shape should be small enough that the BFS only takes

[1] A disjoint set maintains a set of trees where every element is a tree node that has a parent pointer. All the vertices in a tree are considered in the same set. The tree root is the representative of the corresponding set.

an insignificant amount of time. Figure 4.7(b) illustrates an example. When the wire is moved away, a template shape was split into two separate template shapes. The connected components and the template costs need to be re-computed via BFS. The delta cost is:

$$\begin{aligned} \Delta cost = & (1+1-4)w_s && (\Delta size) \\ & + (1-3)w_p && (\Delta diagonal) \end{aligned}$$

The above algorithm is summarized in Algorithm 2.

Algorithm 2 Update the solution and delta cost after wire permutation.

```
1:  procedure IncrementalCost(w, t)                ▷ w: Wire. t: track
2:      cost ← 0, s ← original track where w is at t
3:      for all Contact c ∈ w do                                  ▷ Case 1
4:          C_t ← list of contacts that conflict with c at t
5:          for all Contact d ∈ C_t do
6:              FindRoot(d) ← c          ▷ FindRoot is a routine of disjoint set
7:              cost ← cost + w_p                ▷ Increase peanut-shaped count
8:          end for
9:          S ← list of template shapes where contact in C belongs to
10:         T_new ← Merge S with c
11:         cost ← cost + w_c
12:         T_split ← template where c belongs to at s                ▷ Case 2
13:         cost ← cost − cost(T_split)
14:         C_s ← list of conflict contact with c at s
15:         for all Contact c ∈ C_s do
16:             L ← BFS                  ▷ Recompute connected component
17:             FindRoot(L) ← c                          ▷ Set the new root
18:             cost ← cost + cost(L) ▷ Recompute cost for the split template
19:         end for
20:     end for
21:     return cost
22: end procedure
```

4.4 DSA Cut Redistribution Problem

Given a 1D layout that has n rows (tracks) with a set of cuts that cut the tracks into real and dummy wires, cut spacing rules and a set of template

patterns, move the cuts such that the cuts can be patterned by DSA. Meanwhile, the overall movements of the cuts should be minimized. In the example in Figure 4.5, the layout contains three tracks and there are ten cuts. In this work, we assume that the layout is gridded. The grid size is the same as a cut. The cuts have uniform size.

The cut redistribution problem is non-trivial. It is different from the 2D pattern matching problem, which involves finding some patterns in a given 2D target. In our case, the cuts must be *moved* to form a template pattern as a match, while the match in 2D pattern matching is fixed. Given a set of cuts, there may be multiple ways to divide them into matches and multiple candidate locations to *place* the match. The constraint that only real wire can be extended further renders the problem more difficult.

4.5 Proposed Method

To generate a valid DSA template mask, we propose a graph-based algorithm to redistribute the original cuts according to a given set of valid DSA templates, and minimize the overall cut movements to limit the performance impact. The proposed algorithm consists of the following steps. Given the cut locations, an undirected conflict graph is constructed and its connected components are found. For each connected component, the cuts are matched to some patterns (templates). After all matches are found in the connected components, the cuts are moved to form the matched template patterns such that the overall movement is minimized. Finally, the positions of the template patterns are optimized to resolve possible conflicts. If there are unresolved conflicts, we select some cuts and merge them, then iterate. In the following, we first give a formal definition of the problem, and then discuss the algorithm in details. The flow of our algorithm is shown in Figure 4.8.

4.5.1 Conflict Graph Construction

In the first stage, we build a conflict graph according to the cut locations. Each vertex in the graph represents a cut in the layout. For each pair of vertices, add an edge between them if they violate the cut spacing rule. After the graph is constructed, the connected components will be computed.

To illustrate the algorithm, we will use the problem instance from Figure 4.5 throughout this section. Figure 4.9 shows its conflict graph and connected components. In the following we will refer to a connected component as a set of vertices. For instance, the left-most connected component is denoted as $\{A, B, F, G, K\}$.

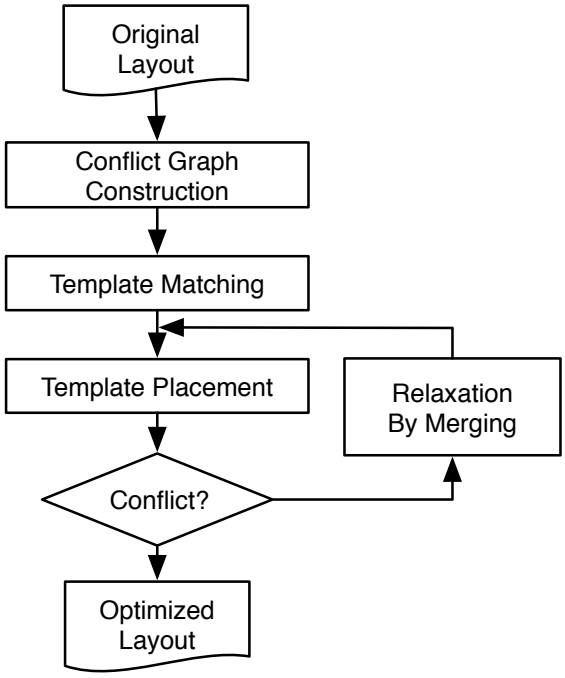


Figure 4.8: Flow chart of the proposed algorithm.

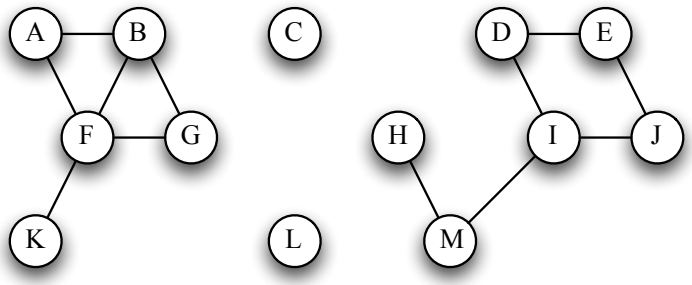


Figure 4.9: Conflict graph and connected components.

4.5.2 Template Matching for Components

In this step, we match the cuts of each component into a set of templates. It will be infeasible to enumerate all the possible matches. We use a scanline

method to determine the matches. First, the cuts are lined up *column by column* from left to right and unmarked. We then use the top-left unmarked cut as the top-left cut in a match. For each template that can be matched, remove the matched cuts and re-lineup the remaining cuts. Note that if there is no match for this cut, we mark it as unmatched and move to the next cut. We iterate the above until all the cuts are either matched or marked. The above scanning can also be done from the bottom-left cut, and the line-up can be from right to left. Thus, there are four possible 'passes': l-to-r and r-to-l directions, each with two cuts to start with. We refer to the collection of matches in a pass as a *match set*. We choose the best match set in one of the passes according to the following precedence:

- Smallest number of marked (unmatched)
- Smallest move cost

where the move cost is defined as the sum of moving distances of the cuts to form the matched template patterns. The rationale of the above is that the smaller the number of marked cuts, the more cuts are grouped, and potentially more space will be saved for the latter stage.

Figure 4.10 shows four match sets obtained from a left-to-right pass that starts from the top-left cut. The match set in Figure 4.10(a) will be chosen since there is a cut unmatched in the other match sets in either match set 4.10(b) or 4.10(c), and it has a smaller move cost than 4.10(d).

We also need to note the following:

1. For the unmatched cuts, we can view each of them as matched to a single-cut template. This is shown in Figure 4.10(b) and Figure 4.10(c).
2. The match set needs to be *feasible*. For example, the match shown in Figure 4.10(b) is infeasible since it is impossible to move the cuts A, B, F and G to form the 2x2 template, unless we shorten the real wires.

Similarly, the match sets for connected component $\{D, E, H, I, J, M\}$ are shown in Figure 4.11. They are computed for the top-right cut from right to left. The match set in Figure 4.11(a) is best as all the cuts are matched. The steps to match are also illustrated. First, the four cuts D, E, I and J are matched into a 4-cuts template. They are then removed from the line-ups. Since this is a right to left line-up, cut H is pushed to the same column as

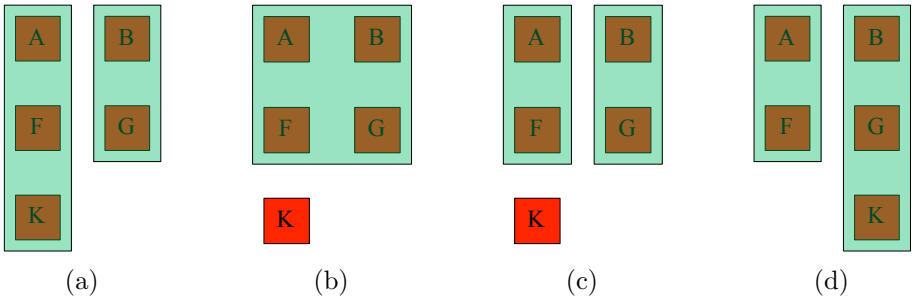


Figure 4.10: Match sets of connected component $\{A, B, F, G, K\}$.

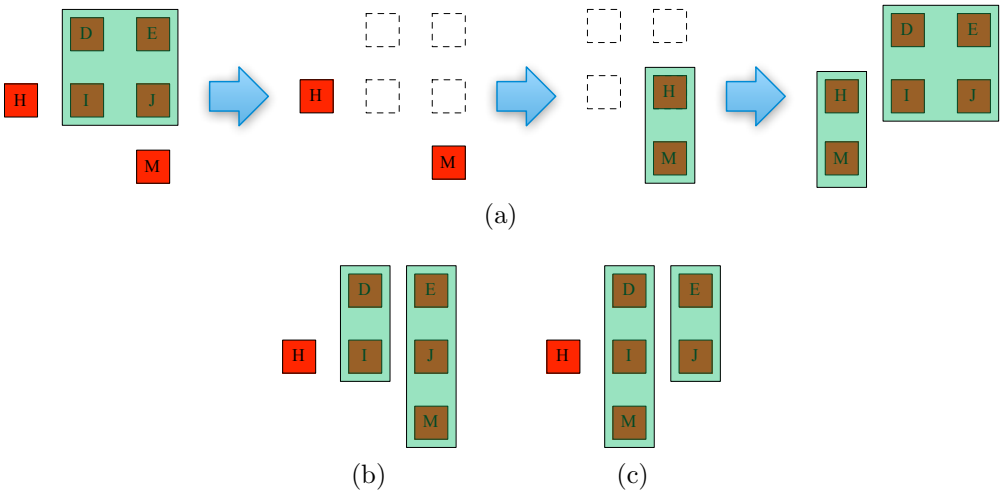


Figure 4.11: Match sets of connected component $\{D, E, H, I, J, M\}$.

M after re-lineup. The final effective match set is shown at the end. Some other match sets of the same component are shown in Figure 4.11(b) and Figure 4.11(c).

Since the types of passes are fixed and the templates are bounded, this step can be done linearly to the number of cuts in the component.

4.5.3 Conflict Pairs

We introduce the concept of *conflict pairs* as a technique to speed up the template matching step. A conflict pair is defined as a pair of cuts that cannot be in a template simultaneously. The conflict pairs of the given layout and template library can be determined as follows. Clearly, only the real wire can be extended. If a real wire is defined by two cuts A and B, they cannot be shifted towards each other to form a template, for otherwise the real wire is

shortened. This condition holds when the distance between two cuts in the same row in any template is smaller than the length of the real wire. Thus, we can pre-process and examine all the cut pairs that define a real wire, and mark them as conflict pairs accordingly. During template matching, we immediately know a template is infeasible whenever there is a conflict pair included. In our example, there are two conflict pairs: {A, B} and {F, G}. Thus, the matching in Figure 4.10(b) is invalid since both pairs are inside the same template. Pseudo-code of template matching is shown in Algorithm 3.

Algorithm 3 Template Matching Process

```
1: procedure TemplateMatching(cuts, templates, conflicts, corner)
2:     Line up cuts according to corner
3:     Unmark all elements in cuts
4:     M ← ∅                                   ▷ M is the set of matches
5:     while cuts ≠ ∅ do
6:         T ← match a template to corner of cuts
7:         C ← the matches
8:         if T = ∅ then
9:             Mark the current cut at corner
10:        end if
11:        cuts ← cuts\C
12:        M ← M ∪ T
13:    end while
14:    return M
15: end procedure
```

4.5.4 Template Embedding

After the matching, we need to embed the cuts and form the matched template shapes. For example, the cuts {D, E, I, J} are matched to the 2x2 template. We thus need to move them to form a 2x2 shape. This can be done by computing the common move range of these cuts. A move range is defined as the possible places a cut can be shifted to from its original position. The common move range among a set of cuts is defined similarly, and can be computed by intersecting the move ranges of these cuts, with the relative positions of the cuts in the template taken into consideration.

Clearly, the minimum cost locations to align a match will be at the extreme points, which is either end of their common move range. Furthermore, the

ends of the common move range are always defined by some cut, which cuts the end of a real wire that 'constrains' the common move range most. For example, the right end of common move range of {A, F, K} is defined by cut A, since the left end of the real wire defined by A restricts A from shifting to the right any further. The process is illustrated in Figure 4.12.

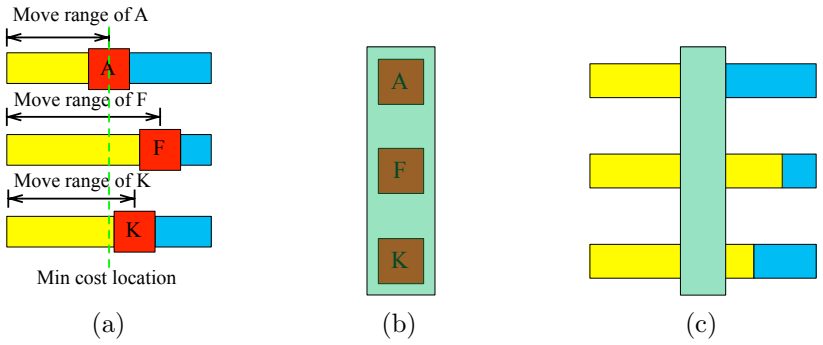
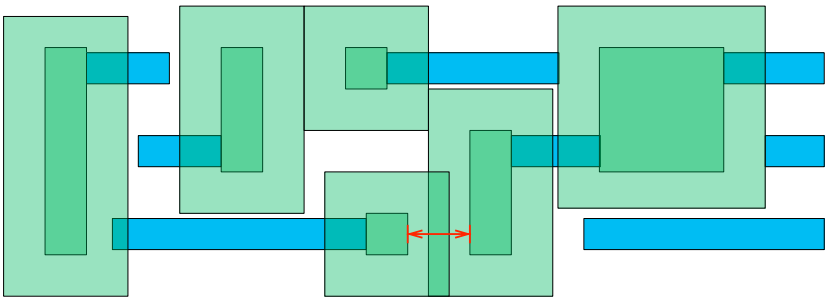


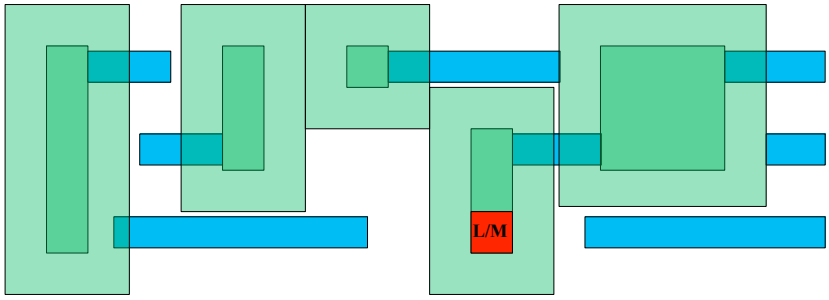
Figure 4.12: Template embedding example.

4.5.5 Legalization and Detailed Placement

When processing a single component, the template matching described in the previous section ignores other connected components. Thus, conflict may exist between matches in different components. This can be formulated as a special *placement* problem. For each shape, we view the out-most cuts as the boundaries of a polygon and expand these boundaries by the value of $r/2$, where r is the value of cut spacing rule. The expanded polygon can be viewed as a 'cell' to be placed along horizontal direction. For each cut in this shape, create a 2-pin 'net' between its original location and the center of the cell. This is similar to the 'fixed port' constraint in the classic placement problem. The real wires can be modeled as fixed cells, while the dummy wires can be viewed as empty space. Note that the relative positions of the shapes are captured in the net wirelength. For example, if originally cell A is to the left of cell B, the cost of placing cell A to the right of cell B must be larger than placing cell A to the left of cell B. Clearly, the feasible placement with minimum wirelength will be the optimal solution to our original problem, if overlapping is not considered. To eliminate the overlapping, we need to *legalize* the obtained placement. Legalization is the process of eliminating all overlaps by perturbing the modules as little as possible. The legalization may



(a) Initial placement.



(b) Placement after merging L and M.

Figure 4.13: Template placement.

perturb the original placement significantly. Thus, we usually use a *detailed placement* step to improve the quality (overall move cost) while maintaining the legality. The template placement problem from the example can be illustrated as in Figure 4.13(a).

Note that this is a special version of the placement problem, and it can be solved easily because: 1) the template can only move horizontally, which means we only need to consider the x-direction; 2) for each template, the move range can be easily computed, since one of the cuts must be aligned to a real wire, and thus can only be moved left or right (single-direction); 3) finally, the move cost is *monotonically non-decreasing*.

After the template embedding step, we already have an initial placement, which may contain overlapping. We can thus apply standard legalization technique to eliminate the overlapping. Here we adopt the idea of a widely used legalization technique called Tetris [45]. In this algorithm, the modules are first sorted by ascending x-coordinate. The modules are then packed to the left one by one to save space for the unplaced modules. We need to consider a special case. Let the current module be P and its predecessor be Q. If they do not share a column, P may still be blocked by Q due to the

expanded rectangle. However, chances are there is enough space to the left of Q for P to fit in. In this circumstance, P should not be blocked.

Note that it is possible that the placement problem does not have any feasible solution. In this case, we need to *merge* the cuts to resolve conflicts. In particular, two adjacent shapes with a dummy wire in between should be chosen to merge. Since the number of shapes is linear with the number of cuts, the possible pairs are also linear. We choose the pair that has the smallest cost after merging. As shown in Figure 4.13(a), the templates {L} and {H, M} conflict with each other, and there is no other way to separate them. We merge them and re-run the placement to resolve the conflict.

The legalization and merging continue until the conflicts converge, *i.e.*, all the conflicts are resolved, except for some remaining conflicts that may be impossible to resolve. In either case, we can still perform detailed placement, which involves optimizing the overall cost. There is already abundant research on the detailed placement. In our work, we optimize the cost by iteratively perturbing the modules and stop when the result converges.

Pseudo-code of template placement is shown in Algorithm 4. The redistribution of the example is shown in Figure 4.5(b).

Algorithm 4 Template Placement Process

```
procedure TemplatePlacement(Matches)
    Modules ← ∅
    for all match ∈ Matches do                          ▷ Cut aligning
        Align cuts in match
        Modules ← Modules match
    end for
    Sorted ← Sort Modules according to their x-coordinate
    for all module ∈ Sorted do                          ▷ Legalization
        Pack module to left-most possible position
        if fail to pack then
            Merge module with its predecessor
        end if
    end for
    Perform detail placement on the packed modules.
end procedure
```

4.6 Experimental Results

The proposed simulated-annealing based contact layer optimization algorithm is implemented in C++ and run on a Linux workstation equipped with 2.8 GHz CPU and 32 GB RAM. We converted the Nangate 45 nm library to 1D cells. Some cells are illustrated in Figure 4.14. We further generated a set of benchmarks to evaluate the algorithm. In particular, a number of cells are randomly chosen and concatenated as a standard cell row. Inter-cell connections are randomly inserted between the cells.

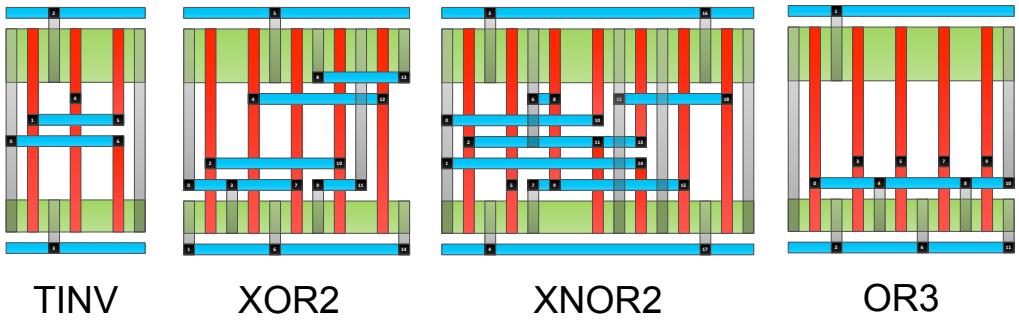


Figure 4.14: 1D cells.

Table 4.1 shows the experimental results of running our implementation on the benchmarks. Five layouts with different sizes are generated for the experiment. The first several columns show some of the statistics of the benchmarks, including number of standard cells (# Cell), number of contact (# Contact) and number of M1 wires (# M1 wires). The initial and final columns show the cost before and after applying the optimization algorithm. We can see that the improvement achieves an average of about 90%. On the other hand, the runtime is affordable even for a layout with 5,000 cells and 60,000 contacts. Note that with enough time for cooling, the solution quality can be improved more.

Table 4.1: Experimental Result of The Proposed Algorithm

Name	# Cell	# Contact	# M1 Wires	Initial	Final	Improve (%)	Runtime (s)
1	10	129	31	45.8	3.5	92.36	0.32
2	100	1262	292	428.3	45.5	89.38	1.18
3	500	6098	1370	1952.7	209.3	89.28	5.49
4	1000	12317	2730	3964.2	401.9	89.86	11.22
5	5000	61081	13341	19627.5	723.41	96.31	67.23

In a 1D standard cell design, the top and bottom tracks are power and ground tracks. Thus, the cell tracks are separated by power and ground tracks, which means cuts between two standard cell rows will not be in the same template. Thus, we focus on solving the problem in a standard cell row. In our design library, there are ten 1D tracks in a row. We generate a set of benchmarks to evaluate our optimization method for cut redistribution, and conduct two sets of experiments.

In the first set of experiments, we generate five benchmarks and run our algorithm on them. The experimental result is shown in Table 4.2. The column 'Initial' means the initial conflict in the layout, and the columns 'After matching', 'After merging' mean the conflict after matching and after iterative merge-and-place phase, respectively. The column 'Cost' means the total cost of moving and merging the cuts. We can see that our algorithm efficiently resolved all the conflicts within seconds.

We compare the effect of different library size on the same set of data in the second experiment. The library sizes we used are full size (four templates shown previously) and half size (only 2-cut and 1-cut). The result is shown in Table 4.3. The columns '4'-'1' refer to the number of templates matched in each test case. We can see that not too many cuts are matched to the 4-cut template, while most of the cuts are still in a single cut template. The result of two-templates is slightly worse than that of the four-templates. However, the conflicts resolved are still huge, due to the effectiveness of the merging and placement technique.

Table 4.2: DSA Redistribution Results

# Cuts	# Conflicts			Cost	Runtime (s)
	Initial	After matching	After merging		
50	72	65	0	74	0.1248
100	115	94	0	226	0.1716
500	703	609	0	1012	1.608
1000	1377	1138	0	2240	2.73
2000	2834	2421	0	4411	15.4783

Table 4.3: Different Library Size Comparison

# Cuts	# Conflict	4 Templates						2 Templates			
		4	3	2	1	merge	cost	2	1	merge	cost
50	72	0	2	2	40	0	74	4	40	1	88
100	115	0	6	8	63	3	226	12	65	11	364
500	703	3	24	34	338	10	1012	65	340	30	1330
1000	1377	8	70	63	607	25	2240	134	656	76	2988
2000	2834	15	125	103	1306	53	4411	287	1298	128	5534

4.7 Conclusion

DSA is becoming increasingly attractive in patterning contacts, vias and cuts. Meanwhile, the special property of DSA process presents to the EDA community unique challenges. In this chapter, we discussed the design-technology co-optimization for DSA in order to better facilitate the advanced process and conquer design challenges. We summarized recent advancements in layout optimization for DSA and explored contact layer and cut layer optimization problems. To model the cost of the variations in different templates, we proposed a cost function for the guiding templates. For the contact layer optimization problem, we proposed a simulated-annealing based scheme. For the cut layer optimization problem, we presented a two-stage algorithm that groups cuts into templates and iteratively places them to obtain a valid layout for DSA process. The experimental results showed that our algorithm efficiently optimizes the contact layer, where the average cost improvement can achieve up to 60%; our algorithm for cut redistribution can effectively redistribute the cuts and resolve conflicts to a great extent, while the runtime is efficiently fast. To conclude, our proposed work demonstrates the promise of utilizing DSA in the 7 nm technology node. Design automation algorithms are expected to play an important role in the next generation EDA tools that incorporate the DSA design flow.

CHAPTER 5

CONTACT LAYER DECOMPOSITION FOR DSA-MP COMPLEMENTARY LITHOGRAPHY

5.1 Introduction

As multiple patterning lithography (MPL) remains the top next-generation lithography candidate with 193 nm immersion (193i) lithography, mitigating the cost incurred by multiple patterning becomes a critical issue [46]. Also, as the EUV technique has been continually delayed, when it really comes out, it will be most likely to adopt double patterning technique to handle the challenges in 7nm technology node. Recently, block copolymer directed self-assembly (DSA) has been receiving much attention with high throughput and low cost [6]–[8], [37]. In DSA process, uniformly sized dense patterns (e.g. contacts, vias, cuts, etc.) are in favor, and guiding templates under DUV or EUV under technology are needed to help regulate the annealing process [10], [12].

To guide the formation of various contact combinations, different guiding templates must be used. However, complex guiding templates may introduce large overlays and the intended contacts may not be patterned correctly. As a result, only those templates that can reliably produce the intended patterns should be used, and we refer to such templates as *feasible templates*. Examples of feasible templates are shown in Figure 5.1. In practice, smaller templates are preferred due to their better variation control, and thus they can be considered having smaller 'costs'.

Because DSA is currently aiming at 7 nm technology, where the guiding template generation needs either double patterning EUV or multiple patterning DUV process, by incorporating DSA into the multiple patterning process, it is possible to reduce the number of masks and achieve a cost-effective solution [47]. Furthermore, it can split the contact patterns into simpler and thus more favorable guiding templates. In this combination,

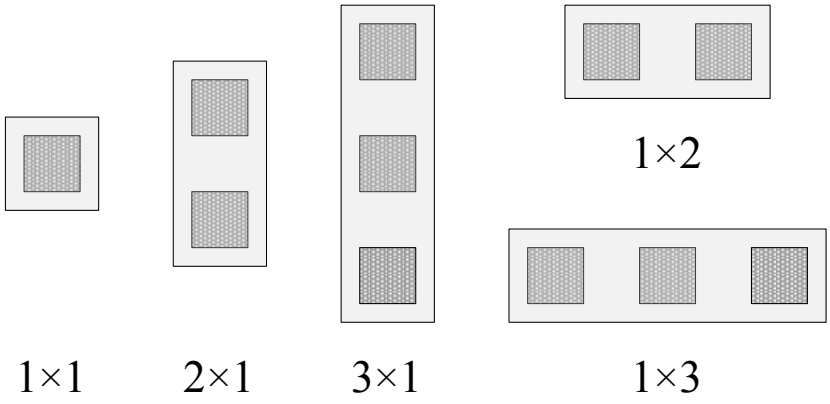


Figure 5.1: DSA templates.

DSA guiding templates are printed via multiple patterning, and the contacts are formed via the DSA process. We refer to this approach as *DSA-MP complementary lithography*. As shown in Figure 5.2, it is possible to use triple patterning lithography (TPL) with DSA for a layout that would otherwise require quadruple patterning or very complex DSA template. On the other hand, by using TPL or DSA alone, the layout cannot be patterned.

For any multiple patterning process, a key issue is the decomposition of the layout into different masks. Decomposition techniques have been proposed for different kinds of MPL [26], [34], [48] and complementary lithography [49]–[51]. Due to the unique characteristics of the DSA process, the DSA-MP complementary scheme faces its own challenges. Existing works on MPL decomposition usually rely on a graph-coloring algorithm. However, the decomposition problem in general 2D layout becomes intractable when the mask number, which translates to the chromatic number of the conflict graph, is three or more. On the other hand, the combinatorial nature of choosing different guiding templates to pattern the layout is also non-trivial even without addressing the template costs. Thus, considering both coloring and DSA grouping simultaneously is very challenging. To enable DSA with multi-patterning technique, the development of corresponding design automation methods is critical.

So far, EDA works for DSA mostly focused on layout optimization [9], [39], [40], [52] and lithography verification [53]–[55], without considering the potential of multiple patterning. A recent work by Badr *et al.* [56] explored using sequential approaches that consider the MP and DSA as two indepen-

dent steps. Their results showed that both of the approaches may fail to find a solution, even for very simple cases. It motivates us to address the constraints in DSA and MP simultaneously and develop effective algorithms for the problem, which is the main theme of this work.

In this work, we study the contact layer decomposition problem for standard cell based layouts with DSA-MP complementary lithography. We will denote the problem as *DSA-MP decomposition* in the remainder of the chapter. As the first step to the study, we propose two heuristics that perform 'color-first' and 'group-first' decomposition for the problem, and we propose an algorithm that decomposes a standard cell row optimally in polynomial-time. Our contributions are as follows:

- We propose heuristic-based approaches that can be used as baseline methods for comparing more advanced algorithms.
- Our decomposition algorithm solves the DSA-MP decomposition problem for a standard cell row optimally in polynomial time. Particularly, a minimal cost decomposition with no coloring conflict can be found efficiently if one exists.
- We conduct extensive experiments to compare the proposed methods and demonstrate the effectiveness of the proposed algorithms. The results show that our optimal algorithm is effective and efficient. This suggests that DSA-MP complementary lithography is a viable and promising approach for the sub 10 nm technology node.

The remainder of this chapter is organized as follows. Section 5.2 discusses preliminaries and formally formulates the DSA-MP decomposition problem. In Section 5.3, we present an iterative color-first and a group-first decomposition algorithm as baseline methods for the problem. The optimal algorithm for standard cell rows is introduced thereafter. Section 5.4 presents experimental results and comparisons of the proposed algorithms. Finally, Section 5.5 concludes this chapter. This work was published in [57].

5.2 Problem Formulation

In standard cell based designs, a set of predefined standard cells is given in a library. The standard cells usually implement basic logic elements such as 2-input NAND gate or a D-flop. The standard cells share the characteristic that all of them have the same height, and two power tracks run through the top and bottom of the cells. The reusability and ease of use make it a very popular design strategy in circuit design. As a result, we will focus on standard cell based layouts in this chapter. Furthermore, we assume the contacts are of *unit size* and *on grid*, which is a reasonable setting for the current 1D style designs. Finally, we use *min_dist* to denote the minimum distance design rule.

As discussed in [7], [47], the qualities of the guiding templates are usually positively correlated with the size. 1D templates are preferred over 2D as the diagonal contact pairs have irregular pitch, in contrast to the natural pitch of the DSA process. To model the preference among the templates, we can assign a 'cost' to them. For instance, smaller templates have smaller cost, and 1D templates have costs smaller than 2D templates.

While stitching technique can be used to solve those undecomposable layouts, it may also cause severe yield loss due to overlay. In this work, we focus on the complementary effort between DSA and MP, and assume no stitching is used. Furthermore, we follow the same assumption in [56] that the self-assembly happens after all the guiding templates have been printed, and thus all contact holes are generated via the self-assembly process.

While there are multiple different choices for the DSA-MP complementary process, *e.g.*, the DSA and MP process can both be used to pattern the contacts, we assume that all the contacts are patterned by DSA, while the MP process is solely for guiding template printing. In other words, a standalone contact can be viewed as patterned by a single-contact template.

We now define the DSA-MP decomposition problem as follows:

Definition 4 (DSA-MP Decomposition Problem)**.** *Given a standard cell based contact layer layout, a set of feasible DSA templates with costs, and the maximum number of masks allowed, find a decomposition that contains a set of contact groupings and a valid mask assignment such that the total cost is minimized.*

Traditionally, the multi-patterning mask assignment problem is solved using *graph-coloring* technique, where each layout feature is formulated as a vertex in a graph and will be assigned a 'color' to denote which mask it is assigned to. Two features that are within the conflict distance cannot be patterned simultaneously in a single mask, and thus need to be colored differently. For DSA-MP decomposition problem, the concept is generalized such that DSA templates are considered. Specifically, when a set of contacts is grouped using a feasible template, they must share the same color as they are formed by the guiding template. An example is shown in Figure 5.2(c), where contact A and C are grouped in a template, and thus share the same color. When determining conflicts, the template shape will be used instead of the single contacts.

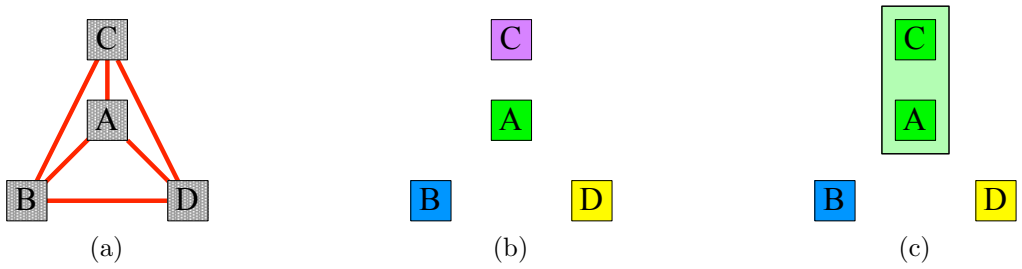


Figure 5.2: Illustration of DSA-MP complementary lithography. Gray mesh denotes the color is unassigned. (a) Conflict graph of four contacts. (b) Solution of 4-coloring. (c) Solution of 3-coloring with DSA, where contacts A and C are grouped into a 2-contact template.

5.3 Contact Layer Decomposition with DSA-MP

As discussed previously, a major difficulty of the problem is to consider both DSA grouping and MP coloring simultaneously. The choices of grouping can be exponentially large when the contacts are clustered closely to each other, which is usually the case in today's aggressive design. Moreover, graph-coloring is intractable when the number of colors is three or more.

It might be tempting to think that the case of two-coloring is easy. When coloring conflict happens, contact grouping can be used to resolve the conflict. In terms of graph theory terminology, grouping two conflicting contacts in the conflict graph is essentially *edge contraction*, which refers to the op-

eration that removes the corresponding vertices' edge and merges the two vertices. All the previous incident edges are preserved and now connect to the new merged vertex. Surprisingly, even though graph two-coloring is trivial, making a graph two-colorable by edge contraction is not. Particularly, making the graph two-colorable is the same as making it *bipartite*. The problem of obtaining a bipartite graph by a sequence of at most k edge contractions in graph G is referred as *bipartite contraction*, which is shown to be NP-complete [58].

Intuitively, the problem relies on finding all odd cycles in G, which is also proven to be NP-complete.

Badr *et al.* [56] proposed two sequential schemes for the problem, which are referred as decomposition-then-grouping and grouping-then-decomposition. Intuitively, one of the operations from multiple patterning and contact grouping is performed first, and then the other is applied. Their results showed that both approaches may not even be able to solve a simple case. Furthermore, template cost is not considered. Based on their simple sequential idea, we propose two iterative schemes, with one of the operations performed as much as possible. They can be viewed a baseline when designing further decomposition algorithms. The following sections outline our algorithms.

5.3.1 Color-first Iterative Decomposition

In color-first decomposition, we first construct the *conflict graph* of the contacts. In the conflict graph, each vertex represents a contact. If the distance between two contacts is smaller than *min_dist*, we refer to them as a pair of conflicting contacts, and an edge is added between the two corresponding vertices. The vertices are then sorted according to a particular order, such as the vertex degree. Colors are assigned to the vertices and an 'EMPTY' color is marked if there is a conflict. For each of the uncolored vertices, we try to find a colored neighbor for which it is possible to group and match a template. If no grouping can be done, flip the color between vertex u with that of its neighbor vertex v that is not in a group. If for some vertex v, it is groupable with its neighbor vertex $w \neq u$, we prefer to choose v for flipping. The flip-and-color procedure continues until a solution is found or the maximum iteration is reached. The rationale is that grouping can poten-

tially resolve the conflict, and the iteration tries to create more groups in the hope that a solution can be found. Pseudo code of the algorithm is shown in Algorithm 5. An example of its execution (assuming triple patterning) is shown in Figure 5.3.[1] Notice that without the iterative flipping procedure, the one-pass decomposition-then-grouping may not be able to find a solution. The color-first method attempts to assign color as much as possible and use less grouping in the hope of minimizing cost while finding a solution.

Algorithm 5 Color-First Decomposition

```
1:  procedure ColorFirstDecomposition
2:      L ← sort vertices according in ascending degree
3:      for all v in L do
4:          color(v) ← next available color c or EMPTY
5:      end for
6:      repeat
7:          for all v where color(v) = EMPTY do
8:              u ← a neighbor of v and can be grouped with v
9:              if u ≠ nil then
10:                 Group u and v
11:                 next v
12:             end if
13:             u ← a neighbor of v and u groupable w/ neighbor w ≠ v
14:             if u = nil then
15:                 u ← a random neighbor of v
16:             end if
17:             Flip color(u), color(v)
18:         end for
19:     until valid solution is found or no improvement can be made
20: end procedure
```

5.3.2 Group-first Iterative Decomposition

Group-first decomposition is similar to color-first, except that we try to group the contacts as much as possible beforehand. To do this, we need to find all possible groupings. We then choose the candidate groupings according to their *size*, until no groupings can be added to the solution anymore. The groups are then colored using a greedy coloring algorithm. Whenever there

[1] Note that A and B (or D) cannot be grouped as this will form an irregular pitch (diagonal) two-hole template, which is not in the library (Figure 4.1).

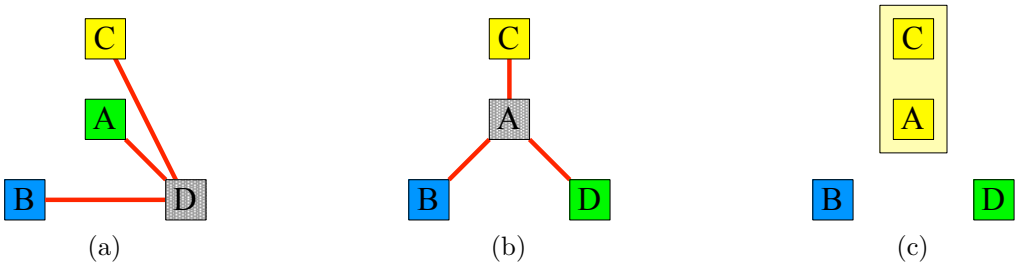
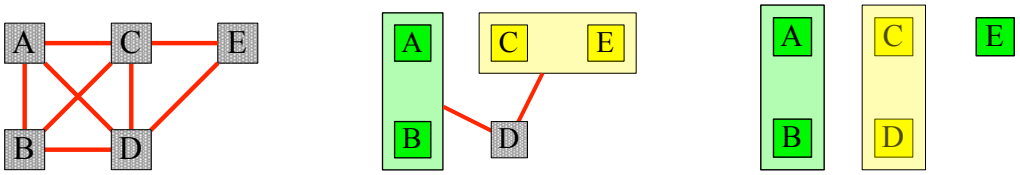


Figure 5.3: (a) Contact D cannot be assigned a color. (b) Flip colors between A and D. (c) Grouping A and C to resolve the conflict.

is a coloring conflict, we choose a candidate grouping for the uncolored vertex, and remove the overlapped grouping to continue the coloring process, until the maximum allowed iteration is reached. Algorithm 6 shows the pseudo-code, and an example using 2-coloring is illustrated in Figure 5.4. The underlying idea of this algorithm is that grouping as much as possible can potentially reduce conflicts and increase the chance of finding a valid coloring solution. However, the total cost may be larger than an optimal solution as there may be excessive grouping.

Algorithm 6

```
1:  procedure GroupFirstDecomposition
2:      G ← a list of all possible groupings sorted according to size
3:      S ← ∅
4:      for g in G do
5:          if g ∩ S = ∅ then
6:              S ← S ∪ g
7:          end if
8:      end for
9:      GreedyColoring(S)
10:     repeat
11:         v ← an uncolored vertex
12:         g ← a grouping such that v ∈ g and v is not grouped
13:         g′ ← a grouping such that g ∈ S and g ∩ g′ ≠ ∅
14:         S ← S \ g′ ∪ g
15:         GreedyColor(S)
16:     until valid solution is found or no improvement can be made
17: end procedure
```



(a) (b) (c)

Figure 5.4: (a) Conflict graph of five contacts. (b) Initial grouping. Contact D cannot be assigned a color as it conflicts with the two groups. (c) Group contact D with C and remove grouping {C, E}. A solution is found.

5.3.3 Optimal Decomposition for Standard Cell Row

As discussed earlier, standard cell based designs present interesting properties, which can be exploited for designing efficient algorithms. We propose an algorithm for a single standard cell row in this section. It contains three stages. In the first stage, all the decompositions for each cell are computed. In the second stage, we construct a *decomposition graph* based on the cell decompositions. Each vertex in this graph corresponds to a cell decomposition. A dummy source and target vertices are added for running the shortest path algorithm in the next stage. Each vertex has a non-negative weight that reflects the groupings cost used in the cell decomposition. An arc will be added between two vertices that do not conflict. The decomposition graph is a directed graph and it is constructed such that any path from the source to the target will correspond to a full decomposition for the row. Finally, a shortest path is found and it is converted back to a full decomposition that has minimum cost. The outline of the algorithm is shown in Figure 5.8(a) on page 91. Since the possible decompositions for cells may be large, we introduce a cell splitting technique to simplify the decomposition graph.

An illustration of the algorithm is shown in Figure 5.5. We denote the jth decomposition solution for cell i as i, j as the vertex id. The details of the algorithm are presented in the following sections.

5.3.4 Decomposition for Standard Cell

The height is fixed among the cells in the library, and the width is reasonably limited as complex cells can be built using basic cells. As a result, we can afford to compute decompositions for each cell by enumeration. Moreover, the

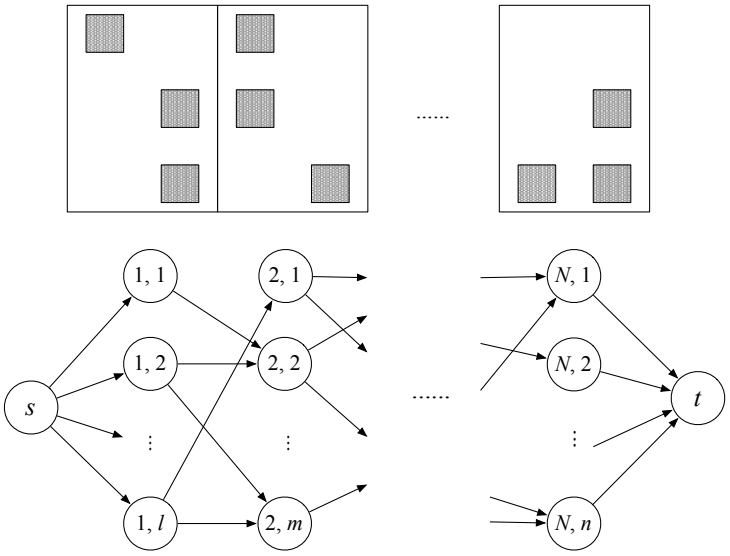


Figure 5.5: Illustration of our row-based algorithm. For each cell, the decomposition solutions are included as a column of vertices in the graph.

Algorithm 7 Algorithm to Decompose A Cell

```
1: procedure DecomposeCell(D)
2:     if all contacts are colored in D then
3:         emit D return
4:     end if
5:     u ← an uncolored contact in D
6:     for all c in colors do
7:         if [ thentry coloring]color(u) ← c does not cause conflict
8:             D.color(u) ← c
9:             DecomposeCell(D)
10:            D.color(u) ← EMPTY
11:        end if
12:        G ← available groupings for u
13:        for all g ∈ G do
14:            if [ thentry grouping]color(g) ← c does not cause conflict
15:                D.groupings ← D.groupings ∪ g
16:                DecomposeCell(D)
17:                D.groupings ← D.groupings \ g
18:            end if
19:        end for
20:    end for
21: end procedure
```

decomposition for the standard cells can be re-used across different designs. As a result, in practice it is beneficial to *precompute* the cell decompositions and store them for later use. Figure 5.6 shows the possible decomposition of three cells. For simplicity and illustration purpose, we use two-coloring in the following. A recursive procedure of generating decompositions for a set of contacts is listed in Algorithm 7. In Section 5.3.6, we introduce a simplification technique such that the total number of vertices and arcs in the decomposition graph can be greatly reduced. Before we discuss this technique, we will first introduce the decomposition graph in the next section.

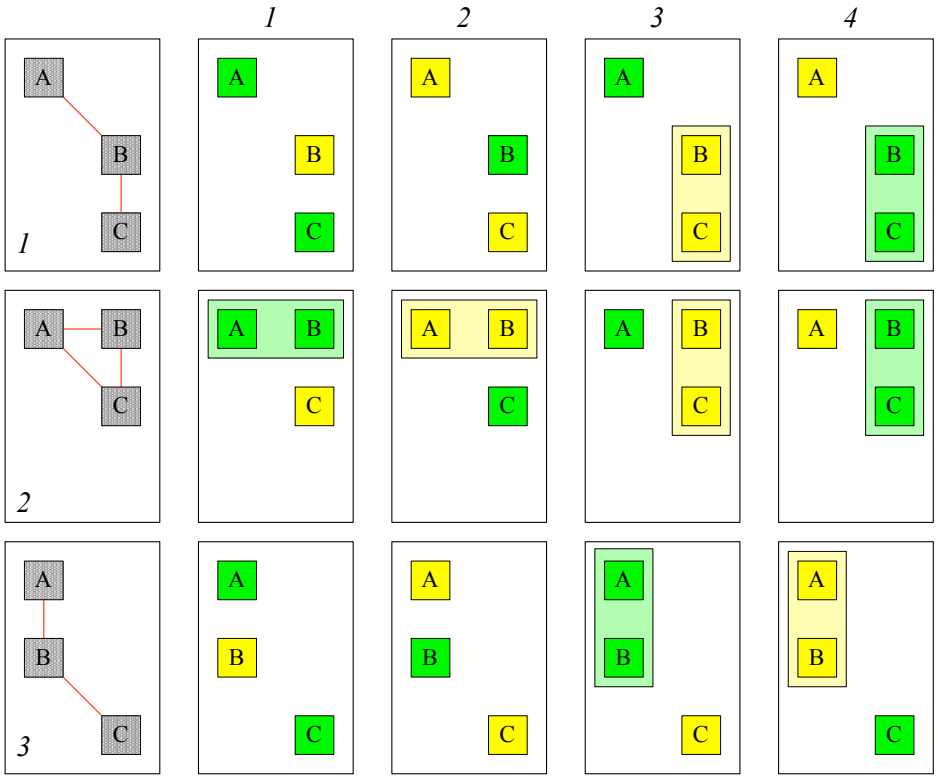


Figure 5.6: Decompositions for three cells.

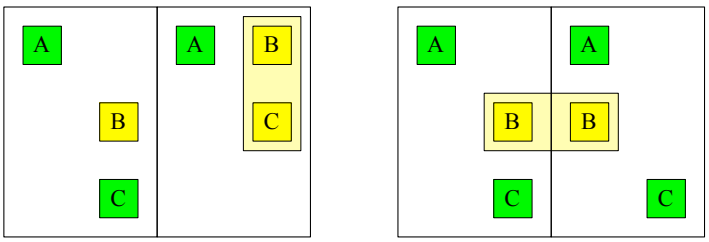
5.3.5 Decomposition Graph

After we have computed the cell decompositions, we can construct a *decompositions graph*. For each cell decomposition, we include a vertex in the graph. The vertex weight represents the total cost of the templates used in its corresponding decomposition. For a pair of vertices from two adjacent cells respectively, we add an arc between them if they are *compatible* or *groupable*.

Suppose u and v are vertices from two adjacent cells, with u from the left cell and v from the right:

- u and v are said to be *compatible* if the color assignment in the u and v is valid between all pairs of conflicting contacts. For example, decomposition (1, 1) is compatible with (2, 3) in Figure 5.6 if we place cell 1 next to cell 2. The compatible decompositions are shown in Figure 5.7(a). As we will run a shortest path algorithm later, we copy the edge weight from the v to the newly added arc as the classic shortest algorithms usually operate on edge weights.

- u and v are said to be *groupable* if the contacts between cells can be grouped, or the existing groupings can be merged as a larger grouping (template). For example, decomposition (1, 1) is compatible with (3, 1) in Figure 5.6 if we place cell 1 next to cell 3. Even though contact B in both of the cells conflict with each other, they can be grouped to make the decompositions compatible again as shown Figure 5.7(b). However, the arc weight should be assigned as $cost_{merge} - cost_{group(u)}$, where $cost_{merge}$ is the cost for the merged groupings, and $cost_{group(u)}$ is the cost of the groups being merged in u, instead of simply copying the weight of v. The idea is to 'cancel out' the grouping cost from u if this arc is chosen in the shortest path.



(a) (b)

Figure 5.7: (a) Compatible decompositions. (b) Groupable decompositions.

We only have to check the compatibility and groupability between adjacent cells, as the *min_dist* will unlikely be larger than minimum cell width (*e.g.*, an inverter). After the arcs are added, we include 1) a dummy source vertex and connect it to the vertices from the leftmost cell in the row; 2) a dummy target vertex and connect the vertices from the rightmost cell to it. Clearly,

running a shortest path algorithm will visit a vertex from each cell exactly once, and the distance from source to target is the total cost of the template. The shortest path corresponds to a full decomposition of the row that has the minimum cost.

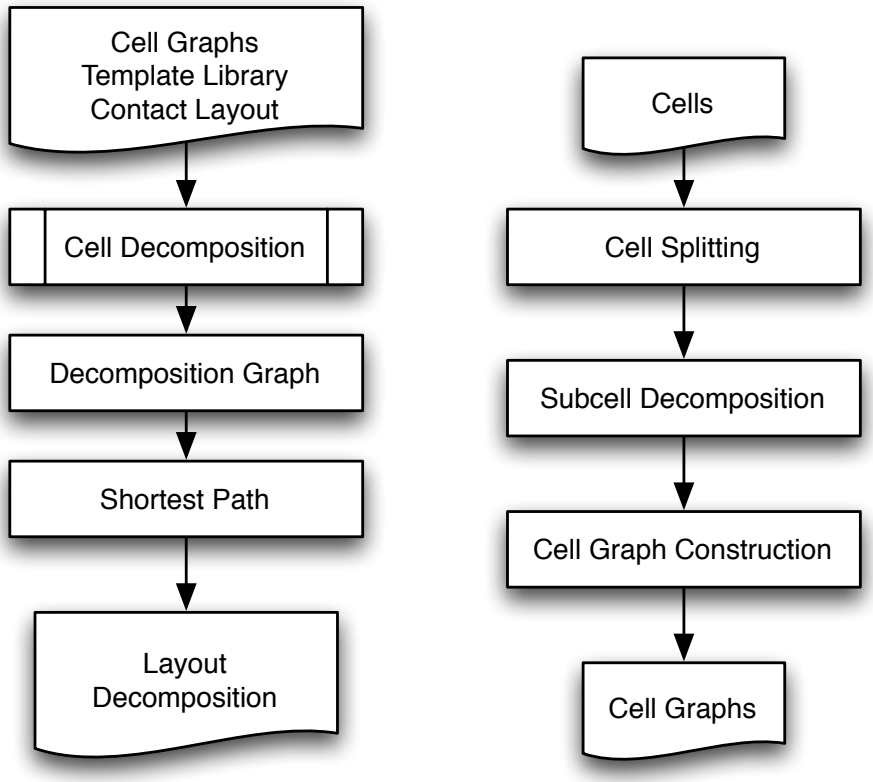
Note that to avoid the 'double grouping' case in which a large template is formed across more than two cells, we need to ensure that the width (height) of the template is always smaller than *min_dist*. This can make sure that a template at the boundary will not reach to the other boundary (so there is at least one grid separation). In practice, the feasible template is usually small (*e.g.*, up to 3 contacts in one direction). We assume the above condition holds for the input cell and template libraries.

5.3.6 Cell Splitting and Simplified Decomposition Graph

Although the cell decompositions are limited in theory, the number of vertices and arcs in the decomposition graph may be prohibitively large when a cell contains many contacts. Large number of tasks may also pose a challenge for the cell decomposition process. To reduce the complexity of the decomposition graph, we can simplify the cell decomposition by *cell splitting*. A flow is shown in Figure 5.8(b).

We observe that the cell interaction is *local*. The contact conflicts between two cells are within a window of width *min_dist*; *i.e.*, if two cells are adjacent, a contact at the right boundary of the cell at left will conflict with a contact in the cell at right at most *min_dist* away from its left boundary. We define the *left (right) boundary contacts* as the contacts that are within *min_dist* from the left (right) boundary of the cell. These are the contacts that can interact with other boundary contacts in the adjacent cells. Thus, to check the boundary condition between two cells, we only need to solve for the decomposition for the boundary contacts and check the compatibility between them. Again, these boundary contacts may also interact with the remaining middle part of the cell, and they can be addressed the same as inter-cell compatibility.

Therefore, we can split a cell into a series of *subcells* and build a decomposition graph for this cell. The widths of subcells need to be *at least min_dist* to guarantee that the decomposition vertices from non-adjacent subcells will not



(a) (b)

Figure 5.8: (a) Outline of the algorithm. (b) Cell splitting and decomposition flow.

interact such that the correctness of the decomposition graph is maintained. By performing cell splitting, the numbers of vertices in the decomposition graph for this cell can be greatly reduced compared to the unsplit version. To see this, assume that there are n contacts in the cell. If the cell is split into k subcells, each subcell will have n/k contacts on average. For two-coloring, the number of possible decompositions is up to $O(2^n)$. On the other hand, cell splitting effectively brings the number of total vertices to $O(k2^{n/k})$. For $n = 12$ and $k = 3$, the numbers of vertices before and after splitting are $2^{12} = 4096$ vs. $3 \cdot 2^4 = 48$.

If there is no conflict between any pair of contacts from two adjacent subcells (*i.e.*, they are independent), every vertex from the first subcell will be compatible and thus connected to every vertex from the second subcell. We can simply include a 'hub' vertex and connect all the vertices from the first subcell to it, and connect it to all the vertices from the second subcell.

For the *independent components* in the non-boundary subcells, they can be excluded from the decomposition graph and handled separately, as they will not interfere with other contacts.

The decomposition graph after cell splitting is referred to as a *simplified decomposition graph*, and we refer to the graphs constructed for the split cells as a *cell graph*. Figure 5.9 illustrates the idea. The cell in the example is split into three subcells. The left boundary contacts include A, B, C and D, and the right boundary contacts include contacts I, J, K and L. There is an independent component {G, H}. Figure 5.9(b) shows the cell graph of this cell. As subcell 2 is independent of subcell 3, a hub vertex is added to connect the vertices from them. It is worth noting that using the cells as splitting units instead of the full row has the practical advantage that the cell-level decomposition graph can be cached for use when running the algorithm for different designs.

A Complete Example

Figure 5.10 illustrates the execution of the algorithm with a simplified decomposition graph. The relative costs of the template should be $cost_{1\times1} < cost_{1\times2} < cost_{1\times3}$. In this work, we use the numbers 0, 5, 8 respectively. The left and right cells each have four solutions. The middle cell is wider and it is split into left and right subcells. Each of the subcells also has four solutions.

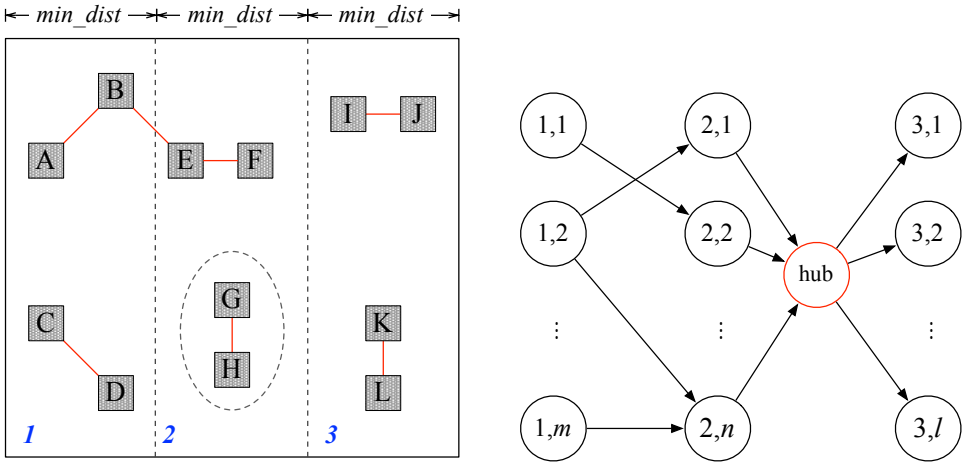


Figure 5.9: (a) A cell is split into three subcells. (b) Simplified decomposition graph for the cell (without source and target).

For the arcs that have weights valued at 8, they are due to the inter-cell grouping as a 1×3 template. A hub node 'h' is added for the components. In the optimal solution, a path with cost 13 is found. It can be converted back to a full composition with two templates. Note that there is more than one optimal solution. This provides flexibility for us to handle the multiple row case.

5.3.7 Complexity Analysis

Supposing the input row contains N cells, each cell is split to k subcells on average, where each of the subcell has n contacts. Let the number of masks (coloring) be m. The number of vertices in the decomposition graph is then $O(m^n kN)$, and the number of arcs is $O(m^{2n} kN)$. Notice that the graph is a directed acyclic graph (DAG); we can use a topological sort based algorithm to find the shortest path that runs in $O(|V|+|E|)$. As a result, the overall running time is $O(m^{2n} kN)$. Since the patterning technology is pre-determined for a given library, and the number of contacts in a cell is bounded by the fixed height and limited width for the subcells, the component m^{2n} becomes a constant. As a result, the algorithm runs linearly to kN. The proposed algorithm is a *fixed-parameter tractable (FPT)* algorithm [59], and the DSA-MP decomposition problem is a FPT problem, which is parameterized by m

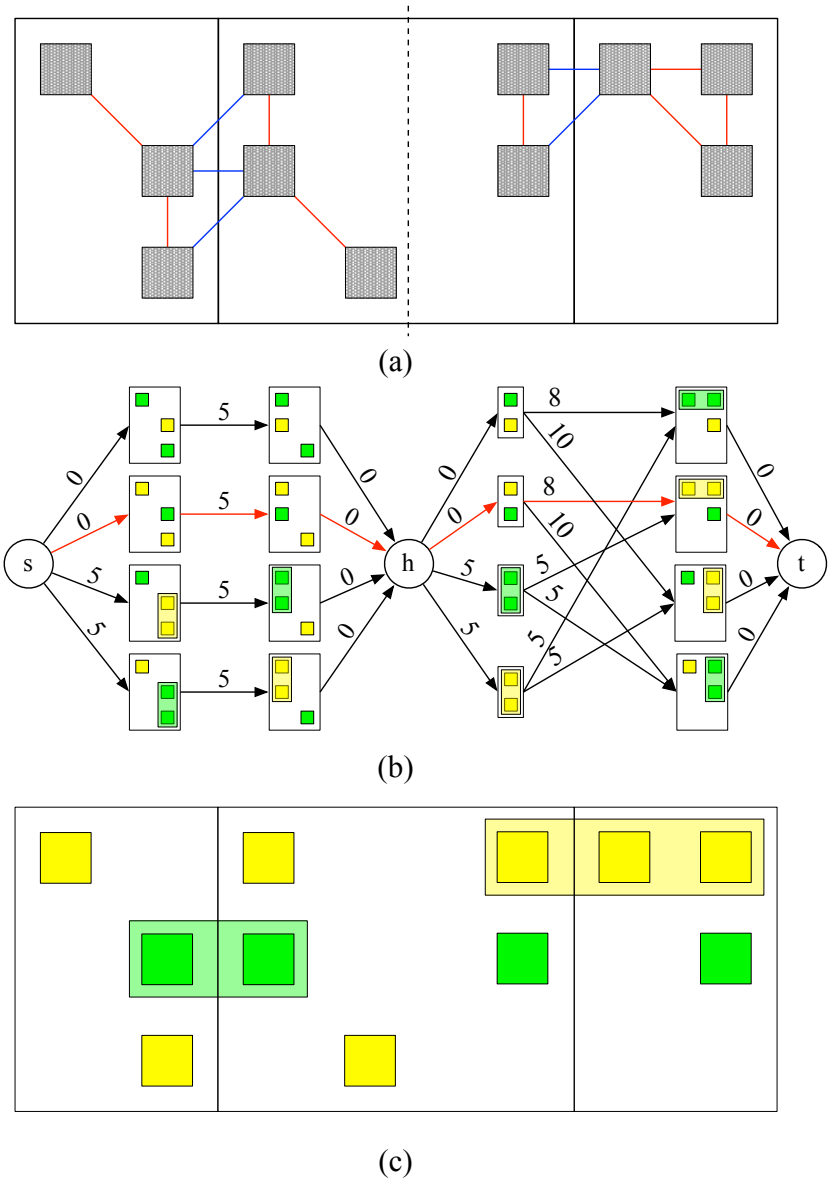


Figure 5.10: (a) A standard cell row that has three cells. The red solid lines denote intra-cell conflicts. The blue solid lines denote inter-cell conflicts. The dashed line divides the cell into left and right subcells. (b) The decomposition graph. The arcs in red denote the shortest path. (c) Optimal decomposition obtained from the shortest path. The full decomposition has a cost 13, and uses a 1×2 and a 1×3 template.

and n.

5.3.8 Handling Multiple Rows

The proposed row-based algorithm can be used to solve multiple rows iteratively. One possible approach is to find the optimal decomposition for the first row first, then use the contacts at the row boundary as constraints to solve for the second row, and repeat. As shown in the previous example, there may be several optimal solutions for a row. Therefore, we can start with each of them to check for a global optimal. Alternatively, one can generate the solutions for all rows, and resolve the local conflicts that happen at row boundaries.

Due to the fact that the power rails run through the top and bottom of the rows, the contacts above the power rails are usually for power accessing and comparatively less than the contacts in between power rails, which are usually introduced by routing. It is possible that the contacts between rows are independent. The inter-row conflicts should also be much sparser than the inter-cell conflicts along the row. As a result, the proposed iterative scheme will produce a small number of conflicts in practice.

5.4 Experimental Results

To compare the performances of the proposed algorithms, we implemented them in C++ programming language and tested in a Linux machine with 2.4 GHz CPU and 34 GB memory. As the benchmarks used in [56] are synthesized using commercial library and tools and they are unavailable to us, we designed two sets of benchmarks for double patterning (DPL) and triple patterning (TPL) based on Nangate 45 nm Open Cell library [60]. The standard cells are rescaled and resized and modified as an on-grid design. Cell rows are generated by randomly selecting the cells in the modified library. The feasible DSA templates in Figure 4.1 are used. We assume that the *min_dist* is 2 grids for DPL with EUV techniques and 3 grids for TPL with 193i technique.

Table 5.1 shows the experimental data for double patterning. After running the algorithm, color-first (CF) has the most conflicts, group-first (GF) the second most, and only the optimal algorithm (OPT) can produce solutions without conflict. We can see that CF uses mostly 1×2 template, and very few 1×3 templates. The reason is that CF prefers coloring. Though

the iterative color flipping helps to find groupings, most of the colors are already assigned and the grouping opportunity is small. The cost of CF is small as it uses fewer groupings, but the remaining conflicts are huge. On the other hand, GF groups as much as possible at the beginning; thus, it produces one order of magnitude fewer conflicts than CF. However, as lots of unnecessary groupings are made, it has a huge cost that is almost twice as large as OPT's, as the groupings used by GF are also twice as numerous as OPT's. In other words, OPT finds a conflict-free decomposition with fewer groupings than GF. In terms of runtime, OPT runs significantly faster than CF and GF. The similar slower runtimes of CF and GF are mainly due to the iterations. In this experiment, OPT outperforms CF and GF in terms of solution quality and runtime.

In the following test, we inserted multiple dummy contacts at the polys in the standard cells. Table 5.2 shows comparison statistics for the TPL case. As seen from the table, a similar conclusion can be made in terms of solution quality. As CF and GF cannot make any improvement after a certain number of iterations, they terminated early and thus used a similar runtime as in the case of DPL. While OPT uses more time than it does in DPL, it finds clean decompositions for all cases.

Figure 5.11 shows the solution obtained by running our implementation on a tiny cell row with five cells. The red solid lines are cell boundaries. The contacts assigned to different masks are colored in green, yellow and blue. We can see that the contacts are successfully decomposed into three masks. Feasible templates are used to group contacts that have coloring conflicts and lead to a clean solution.

Table 5.1: Comparison of the Algorithms (DPL)

	# Cells	# Conflicts*	# Conflicts after†			# Grouping (2, 3, total)			Cost			Runtime (s)		
			CF	GF	OPT	CF	GF	OPT	CF	GF	OPT	CF	GF	OPT
1	50	451	12	2	0	35, 0, 35	91, 16, 107	36, 4, 40	175	583	212	0.012	0.008	0.042
2	100	876	35	5	0	64, 0, 64	163, 33, 196	73, 9, 82	320	1079	437	0.021	0.016	0.064
3	500	4491	231	23	0	368, 0, 368	771, 182, 953	377, 76, 453	1840	5311	2493	0.216	0.213	0.249
4	1000	9258	515	44	0	786, 1, 787	1595, 398, 1993	816, 173, 989	3938	11159	5464	0.776	0.816	0.490
5	1500	14058	815	63	0	1217, 0, 1217	2436, 615, 3051	1255, 275, 1530	6085	17100	8475	1.698	1.783	0.785
6	2000	18664	1043	99	0	1567, 2, 1569	3196, 801, 3997	1632, 351, 1983	7851	22388	10968	2.999	3.173	1.029

* Number of conflicting contact pairs † Number of contacts that are unable to be colored due to conflict
CF: Color-first algorithm GF: Group-first algorithm OPT: Optimal algorithm

Table 5.2: Comparison of the Algorithms (TPL)

	# Cells	# Conflicts	# Conflicts after			# Grouping (2, 3, total)			Cost			Runtime (s)		
			CF	GF	OPT	CF	GF	OPT	CF	GF	OPT	CF	GF	OPT
1	50	1381	64	8	0	52, 0, 52	71, 47, 118	48, 22, 70	260	731	416	0.014	0.009	0.501
2	100	2713	104	13	0	115, 0, 115	126, 99, 225	99, 36, 135	575	1422	783	0.024	0.019	0.888
3	500	13350	550	69	0	506, 9, 515	650, 443, 1093	530, 167, 697	2602	6794	3986	0.248	0.236	4.042
4	1000	26873	1140	122	0	1046, 19, 1065	1377, 867, 2244	1091, 335, 1426	5382	13821	8135	0.886	0.865	7.954
5	1500	40352	1653	163	0	1608, 30, 1638	2094, 1292, 3386	1639, 510, 2149	8280	20806	12275	1.929	1.899	12.161
6	2000	54311	2277	242	0	2058, 39, 2097	2733, 1754, 4487	2200, 696, 2896	10602	27697	16568	3.433	3.376	16.603

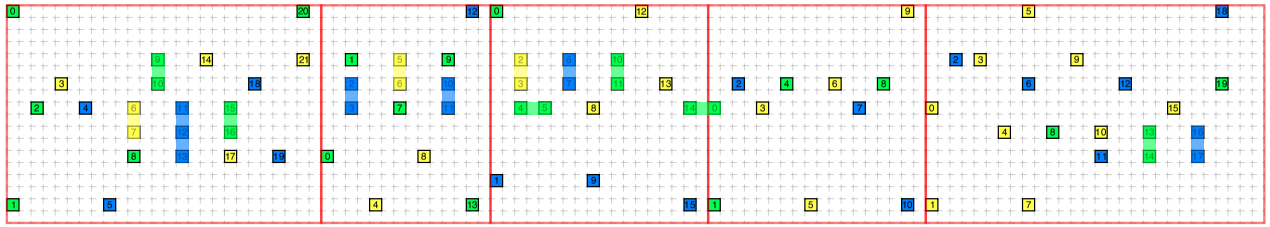


Figure 5.11: DSA-MP decomposition example for TPL.

5.5 Conclusion

As industry today relies heavily on multiple patterning lithography, utilizing alternative lithography provides an attractive means to reduce the manufacturing cost. Directed self-assembly is shown to be a viable candidate to complement with multi-patterning for contact layer patterning. In this work, we discuss the DSA-MP contact layer decomposition problem, the key problem to make DSA-MP complementary lithography practical. We focus the problem on standard cell based layout, and approach the problem in several different ways. A color-first and a group-first algorithm are proposed as baseline methods to study the problem. An optimal algorithm is then proposed to solve the problem in polynomial-time. Our experiments demonstrate that the proposed optimal algorithm can find a conflict-free solution with minimum cost, while both of the color-first and group-first algorithms cannot or use a sub-optimal cost. Furthermore, the efficiency of the optimal algorithm is also superior, as the cell decompositions can be solved and re-used for different designs using this library. This demonstrates the effectiveness of the proposed algorithm for the problem, thus enabling the use of DSA with multi-patterning for circuit manufacturing.

CHAPTER 6

DSA TEMPLATE VERIFICATION

6.1 Introduction

In DSA process, while the contact holes and vias are formulated by the annealing process [10], [12] guided by the guiding templates, those guiding templates are generated by the optical lithography process such as 193 nm immersion lithography, which has a coarser pitch resolution. Like other lithography techniques, the process variation control and proximity correction are the most important issues. The variation in the templates is very likely to affect the final contact. Most critical defects for contact layers are usually introduced by two types of process variations: overlay misalignment and contact size inaccuracy. If contacts are misaligned with metal layers, the effective conducting area is reduced, introducing higher contact resistance or even broken connections in more severe situations. Similarly, inaccurate contact size control may also result in circuit performance variations or even logic errors. Therefore, in order to enable the DSA technology in contact layer printing, it is extremely important for the simulation tools to accurately and efficiently verify the DSA template patterning and hole generation result during the verification phase. We define the templates that pattern the contact holes with undesired variation larger than a threshold value as *DSA hotspots*, or *infeasible templates*.

Although a rigorous DSA simulation can link the guiding template generation process to the DSA pattern formation [10]–[13], the extremely low efficiency makes it impossible to adopt in the full-chip level implementation like DSA-aware OPC and lithography verification. For example, the DSA simulator proposed in [61] consumes an average of 5 minutes to simulate a 2-hole template in 1/2 nm scale. It is time-prohibitive to simulate all such templates in a real circuit.

On the other hand, machine learning (ML) or ML-based compact modeling has demonstrated its effectiveness and efficiency in verification, such as lithography hotspot detection [62]–[67]. In the ML-based approach, statistical models are trained by 'learning' from existing training data, and predict/classify the desired property accurately and efficiently. As a result, machine learning (ML) or ML-based compact modeling is probably the future choice to resolve the efficiency issue in DSA template verification. In this flow, the guiding template can be simulated with the conventional optical lithography simulation tools; then without explicitly simulating the contact/via shapes, we use ML technique to classify whether the template shape is of good or bad quality, the contact pitch, and contact locations. The prediction can subsequently be used as a guideline for the further correction of the guiding template design. To the best of our knowledge, this is the first work to address this problem.

In this chapter, we propose a design automation methodology for DSA template verification and related problems. Our framework includes a DSA model with corresponding ML features (variables), a set of learning algorithms and feature extraction techniques, and a general flow for building classification/prediction models for practical applications. Based on the proposed methodology, we further discuss the *DSA hotspot detection* problem and *contact pitch and location prediction* problem, and propose corresponding solutions.

Hotspot detection and contact pitch/location predication are the key steps in a DSA-aware resolution enhancement flow, which is illustrated in Figure 6.1. Given a target contact layer, the first step is to group adjacent contacts together to generate the layer of target template. Then DSA-aware OPC is performed on the target template layer to create the post-OPC layout. Next, template contours are simulated with 193i lithography models. Based on the simulated contour, DSA verification is performed next to guarantee the robustness and correctness of the DSA hole formation. In particular, hotspot detection can help the designer quickly pinpoint suspicious locations that are potential hotspots. Contact pitch/location prediction can be used to compare with the target contacts and analyze overlay errors. Moreover, we are able to detect the critical edges on the template contours that contribute most to the overlay errors, which provides very important information for template contour correction as well as the DSA-aware OPC process.

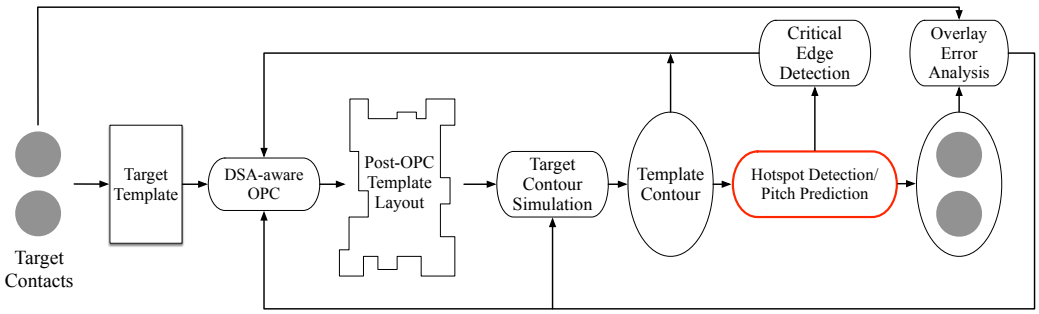


Figure 6.1: DSA-aware resolution enhancement flow.

The rest of this chapter is organized as follows. Section 6.2 discusses related background information, including problem definitions, and gives overview of our proposed methodology and DSA model. Section 6.3 introduces the features we use in the machine learning models. Several machine learning algorithms are briefly introduced in Section 6.4. Section 6.5 presents how we prepare the training and evaluation data for model training. We conducted extensive experiments to test the performances on different features and learning algorithms, and we summarize them in Section 6.6. Finally, conclusions are drawn in Section 6.7. The works presented are published in [54], [55], [68].

6.2 ML-based DSA Verification Flow

6.2.1 Problem Definitions

According to the needs and availability of algorithms and tools, we may encounter different types of problems in DSA verification. For example, people may be interested in knowing whether the printed template shape will eventually generate the desired DSA holes. Without rigorous simulation or prior information, it will be difficult to have a one-stop solution. To tackle this problem, we can adopt a top-down divide-and-conquer scheme by first classifying the templates into known template shapes in the library. This enables us to further examine the feasibility of this template with the tailor-made algorithm, such as whether the pitch sizes of the holes are within feasible range. In other words, the example problem can be solved by levels. The higher level involves determining the type of a template, *e.g.*, whether it

is 2×2 or 1×3. The lower level involves classifying and verifying the quality of the identified template, *i.e.*, whether it will produce a feasible two-hole (1×2) template. In the following, we formally define the above-mentioned problems that can be found in DSA verification.

Definition 5 (DSA Hotspot Detection)**.** *Given a DSA template shape, classify it as hotspot or non-hotspot.*

An application of ML-based DSA Hotspot Detection will be the integration with EDA tools that can help the designers detect potential hotspots on-the-fly during the design process, or guide the EDA layout tool to avoid hotspots.

Definition 6 (Contact Pitch and Location Prediction)**.** *Given a DSA template shape, estimate the contact pitch and locations after the DSA process.*

ML-based efficient prediction of pitch and location will help the DSA resolution enhancement flow iterate much faster.

6.2.2 Proposed Methodology

To address these problems, we propose a machine-learning based DSA verification flow. Based on the flow, we can plug-in different tools (features, learning algorithms etc.) to solve the problems. Our methodology consists of two stages that include various techniques from image processing, computer vision and statistical machine learning as illustrated in Figure 6.2.

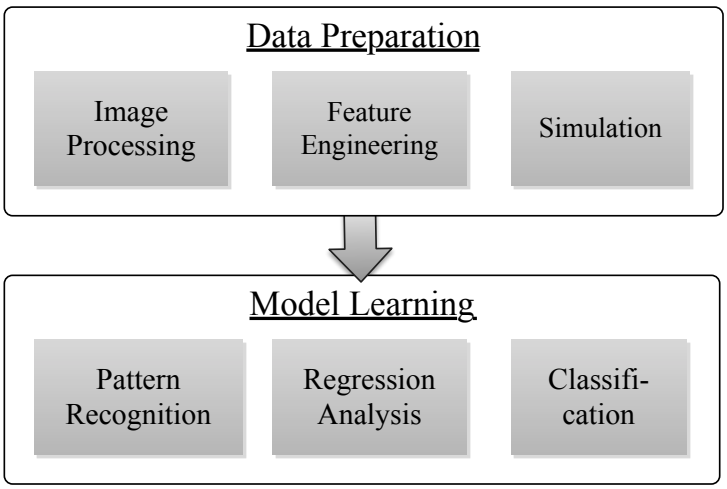
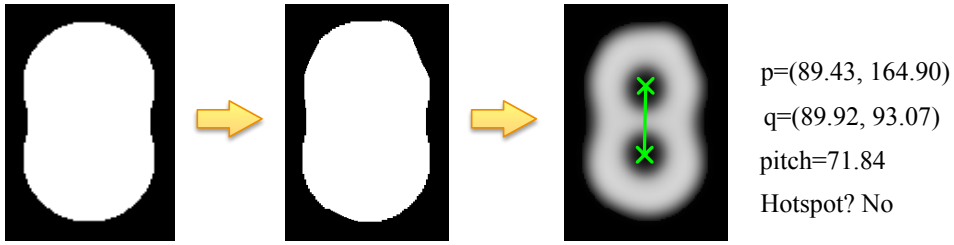


Figure 6.2: DSA verification methodology and techniques.

The first part is data preparation. The starting point of our flow is usually the SEM images of template shapes. Template shapes need to be separately extracted for the later tasks. This involves various image processing and computer vision techniques such as smoothing, resizing and image calibration to prepare high quality template patterns. Next, the template shape information, such as the contour, holes and their pitches, needs to be extracted or measured. Finally, for machine learning related tasks in the second part, features need to be extracted and training data needs to be prepared.

The second part includes various model learning tasks and techniques. Using the data prepared from the first phase, we can build models for different tasks such as template recognition, pitch size estimation and hotspot detection. Statistical techniques such as regression and machine learning can be used for the model fitting.

A simulation-based approach is illustrated in Figure 6.3. Given an original template mask in Figure 6.3(a), a *possible* DSA template will be printed as shown in Figure 6.3(b). The contact information including pitch and location can be obtained after simulation (Figure 6.3(c)). The problem of this approach is that the DSA template printed is susceptible to variations, which may be caused by the process itself or proximity effect by other nearby templates. Thus, each template must be verified individually. Verifying millions of such template shapes in full-chip layout efficiently is a very time-consuming process under this approach.

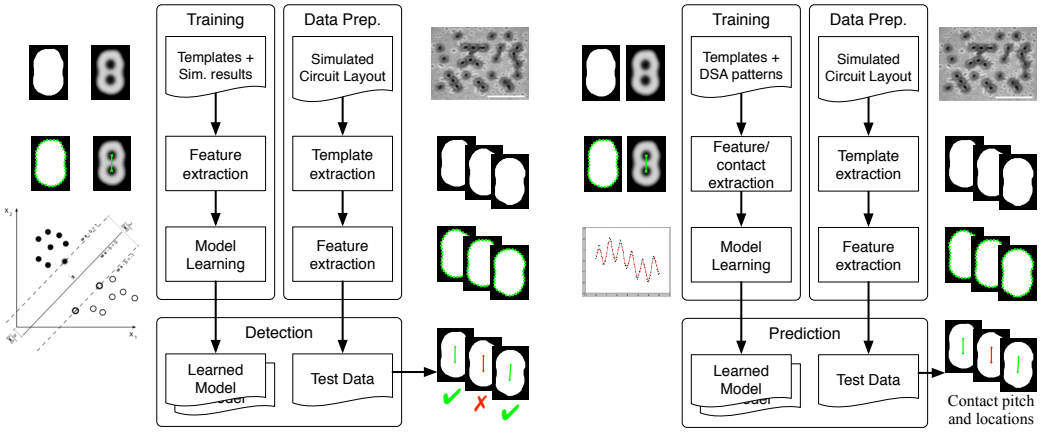


(a) Original mask. (b) DSA template printed. (c) Contact holes after simulation.

Figure 6.3: Simulation-based approach.

Figure 6.4 shows the proposed machine-learning approaches for hotspot detection and contact pitch and location prediction. In the *Training* stage, we learn contact pitch and location regression models from the training data that contains sample DSA templates and simulated DSA patterns. We propose a DSA pattern model for effective learning in the following section. Three

types of features are outlined in Section 6.3. In the Data Preparation stage, the template shape and features will be extracted using techniques similar to those in the training stage. Finally, in the Prediction stage, the trained model will be used to predict the contact pitch and locations.



(a) Hotspot Detection (b) Pitch Prediction

Figure 6.4: Proposed machine-learning based flows.

6.2.3 DSA Template Pattern Modeling

Figure 6.5 shows a geometric model for modeling the template-contact relationship. Under this model, the contact is 'generated' by the confining template shape. To quantify such a relationship, various geometric properties of the template shape can be exploited. For example, if we divide the template perimeter into discrete edges, we can consider each edge to contribute a certain amount to the contact generation. More precisely, we can define *edge sensitivity* w_i as the significance an edge will contribute to the contact. For example, a shift of 1 nm of edge i with large w_i will cause the contact to shift a non-linear amount, while a shift of 1 nm of edge j with small w_j will not affect too much. Let z_i be the geometry information of edge i (*e.g.*, distance from the original mask); then we can have a linear equation to describe the template feasibility f_k for some template k: $f_k = [\mathbf{w} \cdot \mathbf{z_k} - b]$, where $[z]$ is a thresholding function with $[z] = 1$ if $z \geq 0$ or 0 for otherwise, b is a threshold value and $\cdot$ is the dot product of vectors. We can then apply machine learning algorithms to estimate the parameters $\mathbf{w}$ and b.

Similarly, we can model contact location as a function in the form of $(x, y) = f(\mathbf{z})$, where (x, y) denotes contact coordinate, $\mathbf{z} = \{z_1, ..., z_m\}$ and m is the number of edges. In the simplest scenario, (x, y) is independent and $f(\cdot)$ is a linear function. We can then use linear regression to fit a model for prediction. Clearly, it is very unlikely the model is linear, and the edge-based variable may not perfectly reflect the relationship. Nevertheless, the proposed model enables us to quantify and study the problem. Furthermore, the experimental results show that the proposed model and feature sets have high accuracy in predicting the contact pitch and location and thus validate their effectiveness.

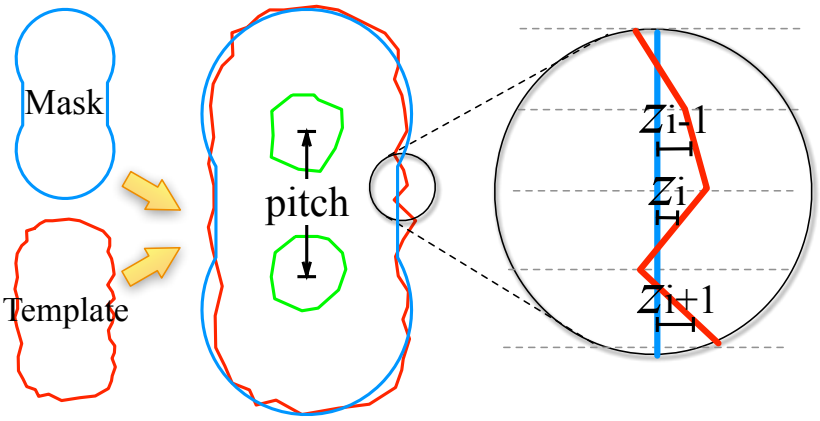


Figure 6.5: Proposed DSA model based on edge sensitivity.

Notice that other parameters may also take effect, such as the DSA process parameters. We assume that these parameters are fixed in a technology library. In reality, the parameter sets should be limited and we can train models for different sets.

Due to the complexity of the DSA process, it will be impractical to build a unified model to verify any possible DSA template. Instead, it is more manageable to build models for different types of templates, *e.g.*, verification models for 1-hole, 2-hole and 3-hole templates. To automatically choose an appropriate model to use, we first need to *identify* the templates. Thus, the verification can be done in different stages and different predictive models can be trained as illustrated in Figure 6.6. Given a set of simulated DSA templates in a layout (Figure 6.6(a)), we first identify the templates (Figure 6.6(b)). The template type is then *recognized* and fed into appropriate models for further verification. To illustrate the idea, we use 2-hole templates

as an example to study the problem, as the two-hole template is the most prevalent case in the layout. The other multiple-hole templates are natural generalizations of the two-hole template and can be studied similarly. For example, to verify a 3-hole template, we can design a model for predicting the locations of three contact holes and two pitch sizes of the top-middle and middle-bottom holes. The same set of features and learning algorithms for training two-hole models can be reused in the other cases.

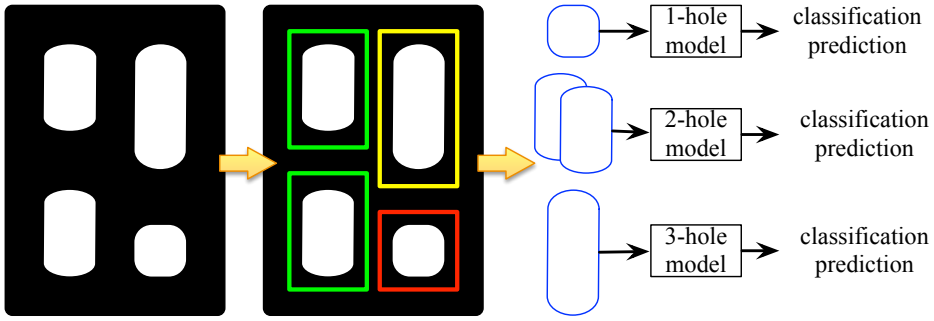


Figure 6.6: Stages in the proposed verification flow and different models.

6.3 Feature Extraction

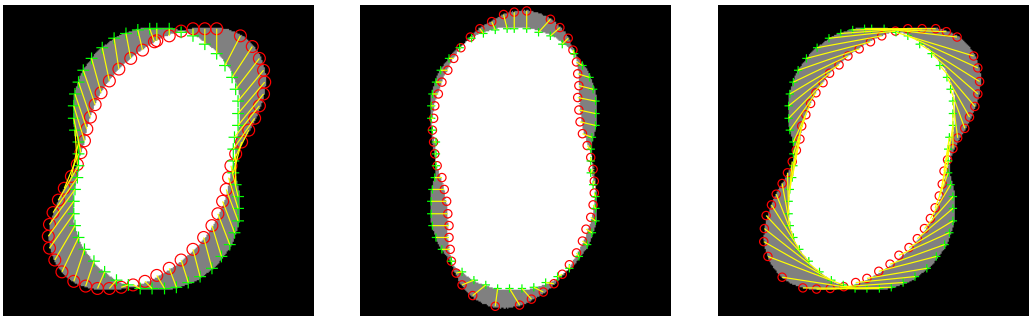
Feature extraction refers to the process of transforming the input data (DSA template) into reduced representations. A feature in the supervised learning context is an individual measurable heuristic property that contributes to the learning. As in any machine learning problem, choosing discriminating and independent features is the key to accurate learning. Based on the proposed DSA model, we present three sets of features and the corresponding extraction algorithms:

- Matched Points feature.
- Point Distance feature.
- Edge Orientation feature.

6.3.1 Matched Points Feature

Given a mask and template shape, we can trace their boundaries and get two sets of boundary points P and Q. We can find a match between P and Q to describe the correlation between the mask and template. More formally, let P_i and Q_j be the ith and jth points in P and Q. A *match* M maps points from P to Q such that $M(i) = j$ matches P_i to Q_j. Clearly, there are exponentially many possible matches, and we are interested in finding a *best* match that correctly captures the correspondence.

We propose an algorithm based on dynamic time-warping (DTW) [69]. DTW has been widely used in signal processing to align and warp time-series data. The basic idea is to use dynamic programming to compute an optimal match between two given sequences such that the overall distance between the matched points is minimized. The time complexity is $O(|P||Q|)$. However, applying the original DTW algorithm to the point sequence is problematic, since the minimum distance match in this case may not be the correct match. For example, if the template is rotated, the correct match will have larger overall distance. An incorrect match generated by applying DTW naively is shown in Figure 6.7(a), where the green '+' markers outline the mask and the red 'o' markers outline the template. We can see that almost all the template points are matched to the mask points in close proximity. We propose a simpler yet more robust solution in the following.



(a) Naive match. (b) Warped match. (c) Correct match.

Figure 6.7: Matched points examples.

The aforementioned problem is caused by the points in Cartesian space, which is rotation-variant. Although there are advanced scale-invariant keypoint features available in the computer vision literature such as scale-invariant feature transform (SIFT), most of them are based on keypoints

such as corners. However, our templates are smooth perimeters in a potentially low resolution image, so it is hard to find meaningful keypoints. To mitigate this problem, we can transform the points to a 'pseudo time-series' by computing the angles between consecutive points (or, derivative of the perimeter). An example of such conversion is given in Figure 6.8(a), where x-axis denotes the point sequence as time series and y-axis denotes the angles. Figure 6.8(b) shows the warped template series. The corresponding point match is shown in Figure 6.7(b). After matching the angles, we convert the match back to the correct point match as shown in Figure 6.7(c).

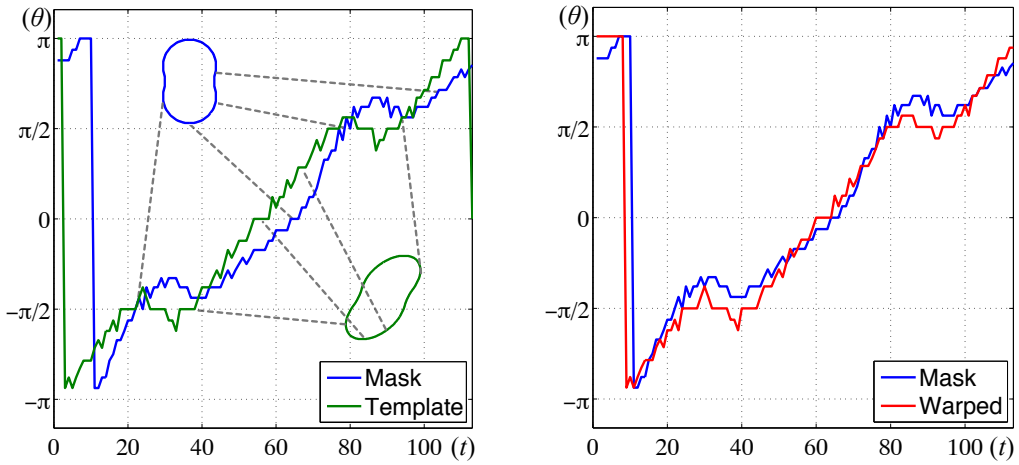


Figure 6.8: (a) Converting mask and template perimeters to time-series. (b) Time-series of mask and warped template.

Note that the DTW algorithm may not match all the points, and thus there will be 'gaps' in the point sequence. We can fill the missing values for the gaps by interpolating the values from the two sides in the point sequence. It is reasonable to assume the unmatched points are on a continuous and smooth curve. An example is illustrated in Figure 6.9, where point i does not have a match, and its corresponding distance value is interpolated from its adjacent points.

6.3.2 Point Distance Feature

Based on the matched points feature, we further propose a point distance feature that captures the template variation. After we find the point match

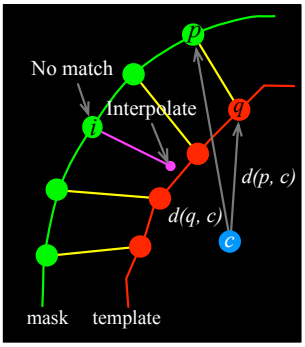


Figure 6.9: Filling missing values.

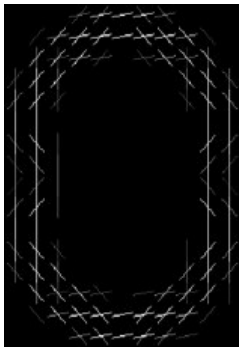


Figure 6.10: HOG features of mask.

and align the template with the mask, we compute the distances for each pair of matched points $p \in P$ and $q \in Q$. We assign a sign to the distance value to reflect the expansion or contraction of a certain template segment. The point distance $d'(p, q)$ of p and q is defined as:

$$d'(p,q) = \begin{cases} d(p,q), & \text{if } d(c,p) \leq d(c,q) \\ -d(p,q), & \text{otherwise} \end{cases} \tag{6.1}$$

where $d(p, q)$ is their Euclidean distance, c is the *center of mass* of the mask. The concept is illustrated in Figure 6.9. Thus, a plus sign denotes the expansion, while a negative sign denotes the contraction.

Even though the point distance feature is derived from the matched points feature, it has the advantage of being half the size of the matched points feature, which is beneficial for the learning algorithm for not only speed but also smaller feature dimension.

6.3.3 Edge Orientation Feature

The orientation of the shape encodes the directional information of the template. A widely used representation is the histogram of oriented gradients (HOG) feature [70]. The idea is to partition the image into uniform bins and compute the local orientation magnitude over eight orientations. The plot of HOG features of the mask is shown in Figure 6.10. Notice how the orientation magnitudes match with the mask.

6.4 Learning Algorithms

For the DSA Hotspot Detection problem, we use *classification algorithms* to classify whether a given input template is hotspot or not. For the Contact Pitch and Location Prediction problem, we use *regression algorithms* to predict the real values. We propose to use artificial neural network (ANN), random forests with bagging (Bagging), boosting and support vector machine/regression (SVM/SVR). We review their ideas in the following sections.

6.4.1 Artificial Neural Network

ANN imitates the human brain by creating a leveled network of neurons. At each level, the neurons take outputs of the previous level as input and output for the next level. Each neuron i can be trained to model some function $f_i : X \to Y$ through *back-propagation* algorithm [71]. An ANN with multiple levels can thus model highly non-linear functions. In our work, we use a three-layer feedforward network shown in Figure 6.11(a). The input layer contains neurons for each input feature, a hidden layer contains a user-defined number of neurons that converts the output from input layer for the last layer, *i.e.*, output layer, which scales its input to the desired response. Again, each layer can be viewed as a function that takes the input from the previous layer and output to the next layer.

The advantage of ANN is that even if we have a lot of input and output data to learn without an idea of what the actual function is, the network is still able to learn it without explicitly specifying the function. Furthermore, ANN is also robust to noisy data, which is our case since the training images may contain noise or imperfect point sampling and point match. A disadvantage is that it suffers from long training time compared to the other methods. Nevertheless, it is usually a minor problem since training a model is an affordable one-time effort given fixed inputs.

6.4.2 Random Forests with Bagging

Random forests [72] is an *ensemble learning* method, which refers to training a 'strong learner' by combining a set of 'weak learners'. A weak learner is a classifier/regressor that is slightly correlated with the true outputs, while a

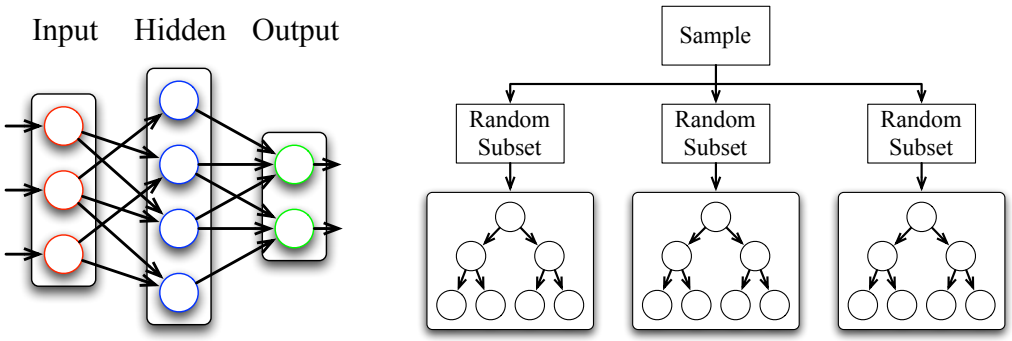


Figure 6.11: (a) Neural network. (b) Random Forest.

strong learner is strongly correlated with the true outputs. Random Forests utilize *decision trees* as the weak learners. In a decision tree, an input is entered from the root node and traverses down to the leaves. At each tree node, the output is split into small subsets according to a feature that provides the best split among all unsplit features. In contrast, random forests choose the optimal split within *a subset of features* at each split. An illustration is shown in Figure 6.11(b).

Bootstrap aggregation (bagging) is a random subspace method that is usually together with random forests. Instead of training from the whole dataset, we generate B bootstrap samples, each by sampling b times with replacement in the original data. The random forests with bagging algorithm is summarized in Algorithm 8. Its major advantage lies in its ability to reduce the correlation and variance for the weak learners and achieve a more stable performance.

6.4.3 Boosting

Boosting is another type of ensemble learning method that also utilizes a set of weak learners such as decision trees. At each tree, the learning samples are weighted in such a way that whenever they are misclassified, they will be weighted more at the next iteration to get more focus from the weak learners. There are various kinds of boosting algorithms. We choose to use AdaBoost [73] for its stable performance.

Algorithm 8 Random Forests With Bagging

Input

Training data $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$

Bagging parameter $k < m$ ▷ m is feature dimension

Training

for $b = 1, \ldots, B$ **do**

$\mathcal{B}_b \leftarrow$ sample size n from $\mathcal{D}$

Train a tree regressor T_b on $\mathcal{B}_b$, where

for each split **do**

Select m features from feature space

Find the best split x_i^* at this dimension subset

Predicting

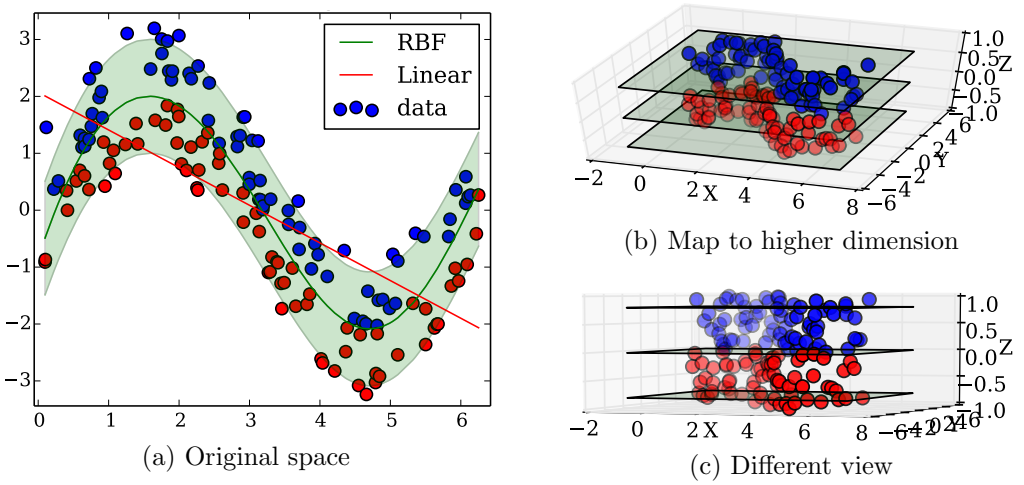
Given an example $\mathbf{x}$, predict $y = \frac{1}{B}\sum_{b=1}^{B} T_b(\mathbf{x})$

6.4.4 Support Vector Machine and Regression

In support vector regression (SVR), each sample $(\mathbf{x}, z)$ can be viewed as a point in $m + 1$ dimensional space, where m is the feature dimension, and the last dimension corresponds to the output value z. For clarity, we discuss the linear case first. SVR finds a *hyperplane* $\mathbf{w} \cdot \mathbf{x} - b = 0$ to maximize the number of data points to be within a distance ε to the hyperplane. The points that are out of the range will be penalized. Pictorially, we can view the hyperplane as a 'band' with width 2ε, and the penalty function forces it to cover as many data points as possible. This type of SVR is called 'ε-SVR'.

To handle the non-linearity, we use the famous *kernel trick* [74] to map the points in space $\mathcal{X}$ to some higher (even infinite) dimensional space $\mathcal{F}$ such that they could be 'covered' in a 'band'. The mapping is referred to as a *kernel function* $\Phi : \mathcal{X} \to \mathcal{F}$. Popular kernels include polynomial and radial basis function (RBF). An example of non-linear SVR is shown in Figure 6.12(a). As we can see, a linear model (red line) will not fit the non-linear data, while a RBF model (green curve) fits them very well. To demonstrate the kernel trick, the points above and below the RBF curve are color-coded to blue and red. We can map these points to a high dimension space as shown in Figure 6.12(b) and 6.12(c), where the mapped points can be 'covered' by a band. We refer the readers to [74] for details.

SVR has the advantage that it is very effective in high dimensional spaces, even when the number of dimensions is greater than the number of samples. The kernel trick and the underlying dual optimization problem make it very



(a) Original space

(b) Map to higher dimension

(c) Different view

Figure 6.12: Example of non-linear support vector regression.

time-efficient in optimization.

6.5 Training and Evaluation Data

In supervised learning, a set of training data is necessary for training a model. The quality of the training data directly impacts the performance of the trained model. Some desired properties include:

- The training data should be accurate and contain as little noise as possible.
- There is enough variance in the feature and output space.
- The size of the training data should be large enough to support the model complexity, *e.g.*, number of features.

Ideally, the training data should be obtained from simulation data or real circuit layout after DSA process. As there is currently no publicly available DSA circuit data, we propose to use a variation model to generate patterns. To model the potential variation that could happen to the template, we propose two operations to vary a given mask image. The first operation includes a subset of affine transformation: scaling, squeezing, rotation and shearing. This operation is applied to the whole image. The second operation is a Gaussian variation to the mask perimeter. In particular, for a perimeter segment,

we apply a Gaussian filter followed by a thresholding to model distortions in the process. Note that these two operations can be mixed to increase the data variety. A plot of the locations of the 1st (upper) contact hole in a 2-hole template in the generated training data is shown in Figure 6.13.

After the templates are generated, we feed them into a simulator based on [61] and obtain DSA contact patterns. To find the contact holes, we threshold the simulation image into a bitmap and run a boundary tracing algorithm to find them. The contact location is then computed as the center-of-mass of the contact hole, and the pitch is computed as the distance between the contact holes. To generate the labels for DSA hotspot, we use a circle detection algorithm based on Hough transform [75] to detect the holes and an automation flow to measure the pitch size, and a threshold to generate the labels.

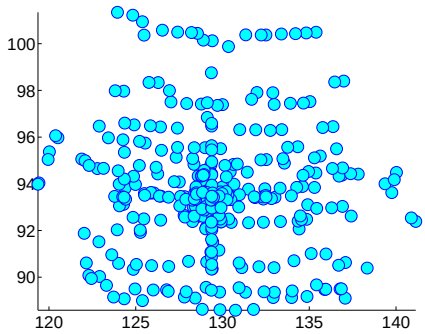


Figure 6.13: Plot of upper contact locations in training data.

6.6 Experimental Results

The proposed flow is implemented in Matlab and the experiments are carried out on a machine with Intel Xeon 2.4 GHz CPU and 32 GB RAM. We use LIBSVM [76] for the SVM/SVR learning, and Matlab Machine Learning Toolbox for boosting, bagging and ANN.

We use the procedure in Section 6.5 to generate a set of 1193 two-hole template samples to train the model. Each of them is derived from a mask image of size 260 × 260 pixels in 0.5 nm scale. We extract the proposed features in these templates. Table 6.1 shows the statistics of feature dimensions. A histogram of the pitches is shown in Figure 6.14. We simulate the templates using a DSA simulator based on self-consistent field theory (SCFT) [61].

Each template costs around 5 minutes in simulation. As we can see, the state-of-the-art simulator is still not ready for the full-chip scale verification. Again, notice that although we are using two-hole patterns to exemplify our proposed method, the method can be generalized for other DSA templates and predict more than one pitch size and numerous hole locations.

Data standardization is important for many machine learning algorithms. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function during the optimization and shadow the effect of other features. Furthermore, numerical stability may also degrade. We subtract means from the features, then scale it by dividing the non-constant features with their standard deviations such that they have zero mean and unit variance to avoid the aforementioned problem. In addition, we remove dimensions that are constant or have less than 10% nonzeros in HOG features due to the sparsity of the template image. This effectively cuts HOG dimensions from 1519 to 428.

Table 6.1: Training Data Statistics

# Samples	μ^*	σ^*	# MP†	# PD‡	# HOG
1193	71.7485	4.3398	1114	557	428

* Mean, Stdev of pitch. † Matched Points. ‡ Point distance.

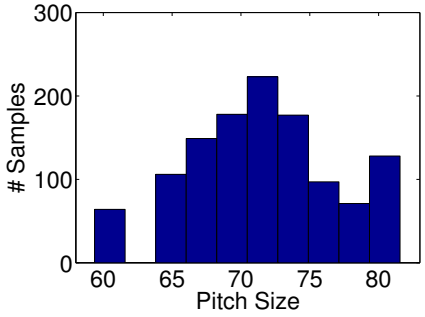


Figure 6.14: Pitch histogram.

6.6.1 DSA Hotspot Detection

For DSA hotspot detection, we compare the performance of different classifiers on different combinations of features. We have the following setups:

- For SVM, we use RBF as its kernel and run grid search to tune the penalty parameter C and kernel variance γ.
- For ANN, we use a two-layer model, where the first layer is an input hidden layer and second layer the output layer. We experimented on 10, 15 and 20 neurons in the hidden layer for each run, and chose the best result.
- For AdaBoost and Bagging, we use Decision Tree as the weak learner, and search for the optimal weak learner size from 100 to 500 at an increment of 50.

Features are mixed and matched for comparison. Table 6.2 shows the accuracy (%) of 10-fold cross-validations for the algorithm and feature combinations, where the accuracy is defined as the ratio of correctly predicted samples over total samples. The total 1193 samples are divided into ten sets, and at each iteration a set (119 samples) will be used as test data while the others (1074 samples) will be used as training data. The average training time is the mean measured among all folds for the largest feature set (MP+PD+HOG). The classifier performances are shown in Table 6.3. In our work, sensitivity is considered more important since we want to detect hotspots (positive) as much as possible, while we can tolerate false negative. In other words, sensitivity reflects the ability to identify hotspots correctly. A negative result in a high sensitivity classifier means a high probability that it is not a hotspot. On the other hand, specificity reflects the classifier's ability to exclude the hotspot correctly.[1] We have the following observations:

1. The average validation times on a set of 61 testing samples are short ($< 1s$) for all the algorithms. Compared to the rigorous simulation runtime that takes around 4 minutes for a single template, it is a huge speed-up.
2. On the other hand, the average training times of ensemble methods (AdaBoost and Bagging) are significantly longer than SVM and ANN. However, our experiment shows that their performances are consistent and more stable than SVM and ANN. Thus, one should consider the trade-off between runtime and accuracy in applications.

[1] To fit in a statistical context, we define 'positive' when the classifier predicts the sample as a 'hotspot'.

3. The detection accuracy rate is over 85% from AdaBoost when all features are combined (dimension = 1761). However, it also has a longer runtime compared to SVM.

4. The proposed sets of features and machine learning do not have very significant differences. It will suffice to interpret that the proposed features can capture the template and hole relationship to a certain extent. When combined together, the best to worst improvement is significant. Each of the learning algorithms is improved around 5-10%.

Table 6.2: Accuracy Comparison of Learning Algorithm and Features

	MP*	PD †	HOG ‡	MP+HOG	MP+HOG	ALL	Train (s)	Test (s)
SVM	89.76	86.43	87.33	90.85	88.02	93.29	0.2012	0.0052
ANN	77.23	76.13	78.55	76.22	75.91	78.71	0.4857	0.0182
AdaBoost	79.04	75.42	79.54	78.86	81.35	86.47	21.1854	0.5680
Bagging	81.35	75.43	80.86	80.34	81.35	82.18	6.9853	0.5996

* Matched Points feature † Point Distance feature ‡ HOG feature

Table 6.3: Performance Comparison of Classifiers

Algorithm	Sensitivity	Specificity	Precision	Accuracy (%)
SVM	0.9548	0.9091	0.9193	93.29
ANN	0.7778	0.7942	0.7409	78.71
AdaBoost	0.8911	0.8453	0.8092	86.47
Bagging	0.7715	0.8614	0.8142	82.18

6.6.2 Contact Pitch and Location Prediction

Table 6.4 shows the performance comparison of the algorithms with different feature sets on predicting the pitch size. The unit of training time is second. As the testing time is within seconds, we do not include them in the table. We use root mean squared error (RMSE) as the metric for performance, which is computed as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2} \tag{6.2}$$

where $\hat{y}_i$ is the predicted value. It gives us an estimate of how 'far away' the average predicted value is from the actual value. The values in the table

Table 6.4: Comparison of Algorithms and Features (RMSE)

Name	MP	Time (s)	PD	Time (s)	HOG	Time (s)	MP+PD	Time (s)	ALL	Time (s)
ANN	0.148	388.969	0.312	251.62	0.17	33.766	0.14	713.38	0.125	446.23
RF	0.292	55.037	0.347	28.044	0.367	18.382	0.329	70.982	0.419	43.964
SVR	0.285	1.656	0.387	1.178	0.233	1.185	0.148	2.577	0.24	2.256

are obtained by running a ten-fold cross-validation for an algorithm and a feature set. Specifically, the data is split into two equal-size sets and at each round, we choose one set as testing data and the rest as training data. We then average the statistics in these ten rounds as the model performance. We have the following parameter selection schemes for different algorithms:

- For bagging, we tune the number of weak learners in the range from 1 to 200. Cross-validation results show that the variance and balance reached a good balance at around 100 weak learners.
- For neural network, we tune the number of neurons in the hidden layers from 1 to 20. The best performance occurs at about 8 neurons.
- For support vector regression, we use a *multilevel grid search* to choose the parameters C (error penalty) and γ (RBF kernel size) from a coarse grid to a fine grid. Our experiments show that $C = 2^6$ and $\gamma = 2^{-7}$ works best. Figure 6.15 shows the contour plot of grid search that finds the optimal parameters.

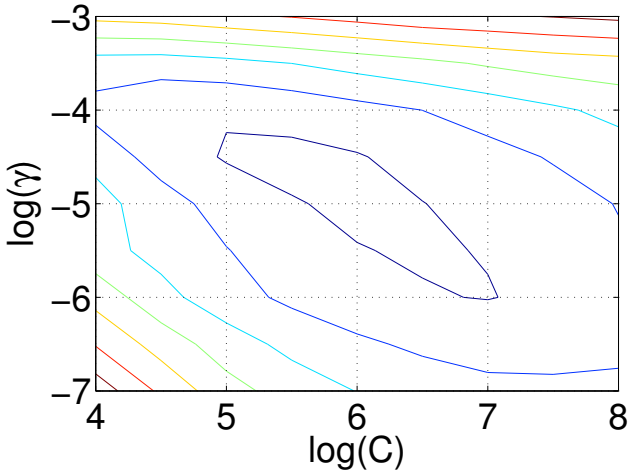


Figure 6.15: Grid search contour plot.

Overall, we can see from the tables that all of the learning algorithms work well and have an average RMSE below 0.5. The prediction accuracy is

Table 6.5: Contact Location Prediction Results (RMSE)

Name	x_1	y_1	x_2	y_2	Mean†	Pitch‡	Time (s)
ANN	0.132	0.145	0.157	0.201	0.158	0.194	64.596
RF	0.476	0.361	0.398	0.351	0.396	0.376	30.297
SVR	0.117	0.137	0.117	0.135	0.126	0.153	0.846

* (x_1, y_1) and (x_2, y_2) denote the 2D coordinates of the contacts.
† Mean error over all the predicted coordinates.
‡ Computed as the Euclidean distance between predicted contact locations.

satisfactory enough, considering that the pitch size granularity is about 0.5, which is decided by the image resolution.

From the tables, we have the following observations:

- ANN outperforms SVR and RF. SVR achieves very close performance to ANN, while RF is slightly behind.
- ANN suffers from a long training time due to high feature dimension. In comparison, SVR is extremely efficient in training, and thus it is very suitable for studying the models for different templates.
- All of the algorithms perform prediction effectively. With a rigorously tuned model that has high accuracy and moderate bias/variance, the machine learning approach is practical for full-chip scale verification.
- Among the single set of features, matched points gives the best performance and training time. While it is usually useful to combine feature sets to improve performance, it is not the case in our experiment. Worse still, the feature dimension and training time are both largely increased.

We have similar results for contact location prediction as shown in Table 6.5. Note that we used an improved MP feature and data set in this experiment, as we will introduce later.

6.6.3 SVR Performance Study and Tuning

We are particularly interested in improving the performance of SVR, as according to the above experiments, SVR has the best balanced performance in terms of accuracy and runtime. We further study its behavior and derive

a model that is more compact and efficient for practical usage. In the following, we split the data into three sets for study: training (60%) set is used for model training, validation (20%) set is used to tune the model parameters and select a final model, and testing (20%) set is used for evaluating the selected model.

Model Selection

The high feature dimension can make the model overly complicated (overfit) and cause high variance. A common approach is to study the dimension- error plot. We uniformly sample dimensions from MP feature, and plot the corresponding training vs. validation error in Figure 6.16. We observe that the cross-validation error hardly improves at dimension size 600. Thus, we choose such a reduced dimension for our model. By doing this, we are effectively *down-sampling* the extracted points from the mask/template perimeter, and we achieve a simpler model that can generalize better.

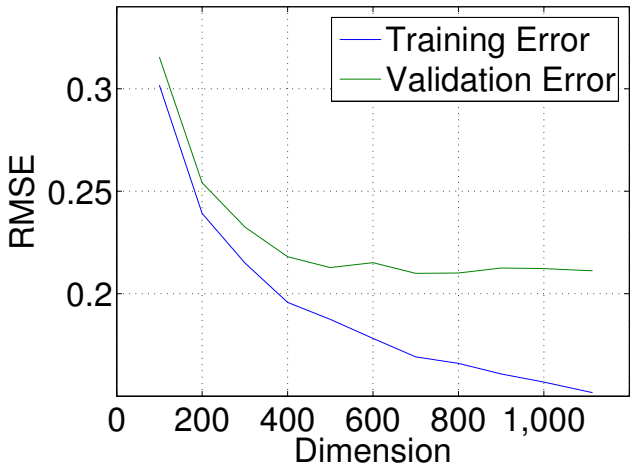


Figure 6.16: Model selection.

Sample Size and Learning Curve

We study the relationship between the error and sample size to understand if there are enough samples for training a highly accurate model. Moreover, a smaller sample size will certainly improve the training time. Figure 6.17 shows the RMSE improvements for the three datasets as the training size

increases. The plot shows that, at around 350 samples, the errors become stable. This indicates the SVR achieves a stable performance with only half the training data.

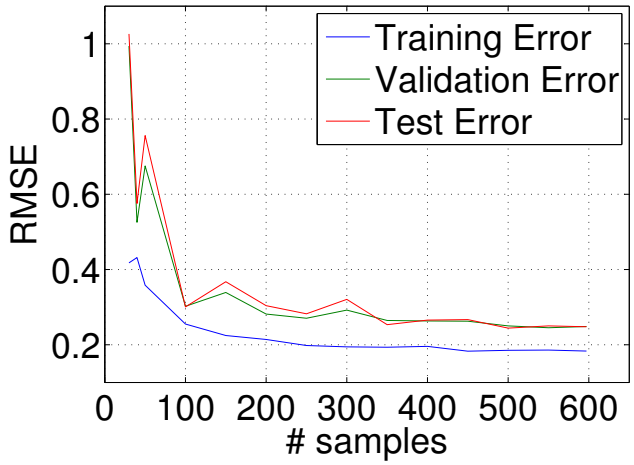


Figure 6.17: Learning curve.

Performance of Tuned Model

We run the simplified model with less training data on the whole dataset and examine its performance. Figure 6.18 shows the error histogram $(\hat{z} - z)$. For all three sets, the predicted values are close to zero error, while the maximum error is bounded within 1.0. Furthermore, we can perform hotspot classification by setting a pitch grid value as the tolerance value. For example, if 0.5 is used, we can see from the histogram that only a small number of examples (37) are misclassified, and the accuracy is up to 97.15%. Figure 6.19 shows a linear regression plot between the actual and predicted values. The closer to 1 the slope is, the more accurate the prediction is. We can see that our tuned model is very close to a perfect line.

Table 6.5 shows the improved performance of the learning algorithms with the simplified models. With the RMSE similar to or even better than the previous results in Table 6.4, the training times are greatly improved. Compared to the time-consuming rigorous simulation methods, our best SVR model achieves near perfect results (RMSE = 0.135 pitch grid). Furthermore, with less than one second of training and predicting runtime overhead, our method is very promising for full-chip scale verification, which is beyond the limit of simulation-based methods. The high accuracy and efficiency

indicate that the machine-learning based approach is very promising as a candidate for DSA verification.

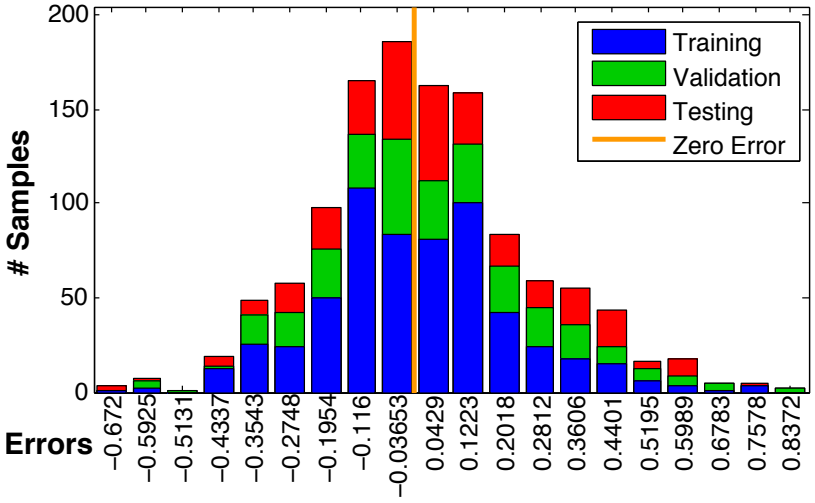


Figure 6.18: Error histogram of predicted values.

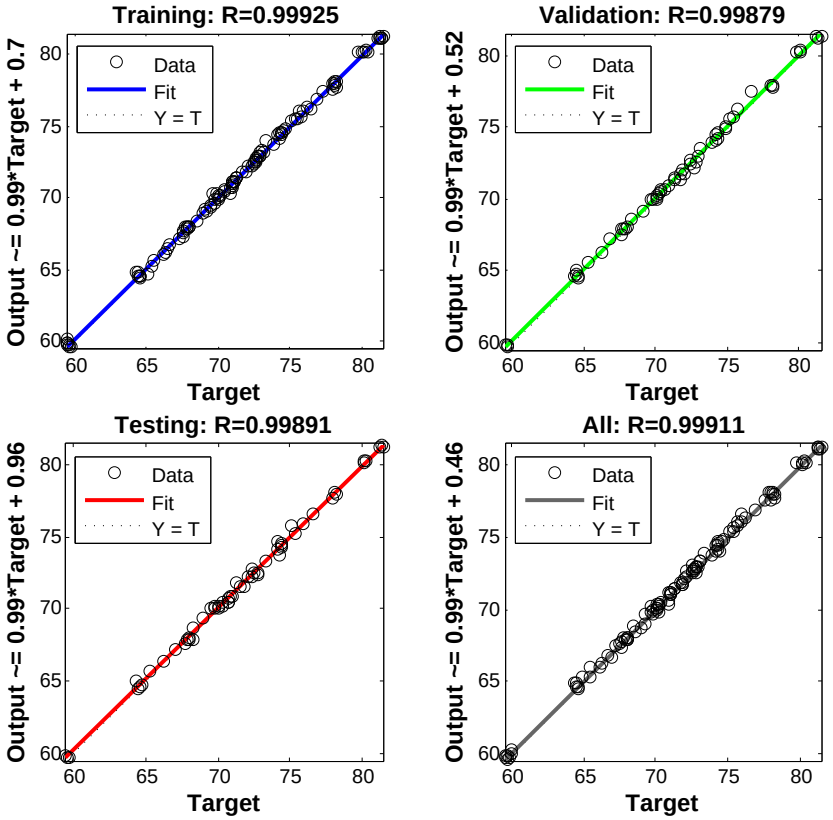


Figure 6.19: Regression plot of the tuned model. Only partial samples are shown due to density.

6.7 Conclusion

As the DSA provides the ability for contact/via patterning and enables it as a candidate for 7 nm node, the EDA world should face the challenge and unleash DSA's potential. This chapter presents a machine learning based approach for DSA verification. We formulated two important problems as DSA hotspot detection and contact pitch and location prediction problems. For the first time these problems are addressed using a machine learning based approach. We propose various sets of feature and learning algorithms to use, and study their performances via extensive experiments. Our experiment showed very promising results for the machine learning based approach. However, as the ML approach is in an exploratory stage, more problems and techniques should be addressed in the future to enable its adoption in industry. For example, how to integrate the flow in a standard verification environment and how to apply online learning algorithms to continuously improve performance in a production environment. These are the future directions that are worth exploring.

CHAPTER 7

CONCLUSIONS

In this dissertation, we have studied design automation algorithms for advanced lithography. We have covered topics on self-aligned double patterning (SADP) and directed self-assembly (DSA). We proposed design automation algorithms to address various issues and constraints from the advanced lithography technologies in order to help the designers better facilitate the potential of these technologies during the design phase.

Like any other double patterning, layout decomposition is the most critical problem to apply SADP in manufacturing. In this work, both general 2D layout and row-based standard cell layout are studied.

In Chapter 2, we studied the layout decomposition problem for SADP in general 2D layout. Although the SADP decomposition for 2D layout is NP-hard in general, we showed that in the case where we disallow overlay, we can derive polynomial-time solutions. We propose a graph-based algorithm to address the problem. To the best of our knowledge, this is the first exact algorithm that solves the problem in polynomial-time. All the previous works are either based on expensive ILP or SAT formulation, or utilize heuristics that cannot guarantee optimality.

In Chapter 3, we studied the layout decomposition problem for SADP in row-based standard cell layout. Due to the special property of standard cell based design, it is usually the case that efficient solutions can be designed for various problems. In our work, we take advantage of the fixed width of standard cells and power tracks on top and bottom of cells and propose an efficient layout decomposition. The efficiency of our method is further demonstrated by the experimental results.

Block copolymer directed self-assembly (DSA) has demonstrated great advantages in patterning contacts/vias for the 7 nm technology node and beyond. The high throughput and low process cost of DSA make it the most promising candidate in patterning tightly pitched dense patterns for

the next-generation lithography. Since DSA is very sensitive to the shapes and distributions of the guiding templates, it is necessary to develop new EDA algorithms and tools to address the patterning rules and constraints of the process.

In Chapter 4, we studied the contact layer and cut layer optimization for DSA. To pattern the contact layer with DSA, we must ensure that all the contacts in the layout require only feasible DSA templates. We propose a simulated-annealing (SA) based scheme to perform full-chip level contact layer optimization. According to the experimental results, the DSA conflicts in the contact layer are reduced by close to 90% on average after applying the proposed optimization algorithm. DSA has proven its success in contact hole patterning, However, the cut layer must be optimized during design to be DSA-friendly. We propose an efficient algorithm to optimize cut layers without hurting the original circuit logic. Our work utilizes a technique called 'line-end extension' to move the cuts and extend the functional wires without changing the original functionality of the circuit. Consequently, the cuts can be redistributed and grouped into valid DSA templates.

In Chapter 5, we studied the decomposition problem for contact layer in row-based standard cell layout with DSA-MP complementary lithography. DSA is currently aiming at 7 nm technology, where the guiding template generation needs either double patterning EUV or multiple patterning DUV process. By incorporating DSA into the multiple patterning process, it is possible to reduce the number of masks and achieve a cost-effective solution. We explored several heuristic-based approaches, and proposed an algorithm that decomposes a standard cell row optimally in polynomial-time. Our experiments show that our algorithm is guaranteed to find a minimum cost solution if one exists, while the heuristic cannot or only finds a sub-optimal solution.

In Chapter 6, we studied the DSA template verification problem and proposed a machine learning based method. Since the DSA technology is very sensitive to the shapes and distributions of patterns, it is necessary for the EDA engines to understand the patterning preferences of the DSA process, such that layout can be optimized to be DSA-friendly, DSA templates will be verified and more preferred patterns will be used. However, traditional rigorous simulation suffers from an extremely slow simulation time. We proposed a machine learning based design automation framework for DSA verifica-

tion. A novel DSA model and a set of features are presented. Following the proposed framework, we formulate the DSA hotspot detection and contact pitch and location prediction problems and address them using the proposed ML-based flow. Extensive experiments are performed to compare the performances of learning algorithms and features. Our experiments showed that we can achieve 93% accuracy for DSA hotspot detection and an average RMSE (root mean square error) of below 0.5 for contact pitch and location prediction.

In conclusion, this dissertation presented two of the most promising advanced lithography techniques, SADP and DSA, and the unique constraints and challenges associated with them. In order to fully release the potential of the new technologies, the EDA industry must embrace the change and follow up with corresponding solutions. Clearly, effective and efficient design automation algorithms are a key challenge and will continue to play a critical role in the future, during which more advanced lithography technologies will emerge.

REFERENCES

[1] H. Zhang and Y. Du, "Self-aligned double patterning decomposition for overlay minimization and hot spot detection," *Digital Automation Conference*, pp. 1–6, Nov. 2010.

[2] H. Zhang, Y. Du, M. D. F. Wong, and R. O. Topaloglu, "Self-aligned double-patterning decomposition for overlay minimization and hot spot detection," in *Proc. DAC*, 2011.

[3] Y. Ban, A. Miloslavsky, K. Lucas, S. Choi, C.-H. Park, and D. Z. Pan, "Layout decomposition of self-aligned double patterning for 2d random logic patterning," in *Proc. SPIE*, Jan. 2011.

[4] Y. Ban, K. Lucas, and D. Z. Pan, "Flexible 2d layout decomposition framework for spacer-type double patterning lithography," in *Proc. DAC*, Jun. 2011, pp. 789–794.

[5] M. Mirsaeedi, J. A. Torres, and M. Anis, "Self-aligned double patterning (sadp) layout decomposition," in *Quality Electronic Design (ISQED), 2011 12th International Symposium on*, IEEE, 2011, pp. 1–7.

[6] H.-S. P. Wong, C. Bencher, H. Yi, X.-Y. Bao, and L.-W. Chang, "Block copolymer directed self-assembly enables sublithographic patterning for device fabrication," in *Proc. SPIE*, vol. 8323, 2012, pp. 832 303–1.

[7] H. Yi, X.-Y. Bao, J. Zhang, R. Tiberio, J. Conway, L.-W. Chang, S. Mitra, and H.-S. P. Wong, "Contact-hole patterning for random logic circuits using block copolymer directed self-assembly," in *Proc. of SPIE*, vol. 8323, 2012, 83230W–1.

[8] H. Yi, X.-Y. Bao, J. Zhang, C. Bencher, L.-W. Chang, X. Chen, R. Tiberio, J. Conway, H. Dai, Y. Chen, *et al.*, "Flexible control of block copolymer directed self-assembly using small, topographical templates: Potential lithography solution for integrated circuit contact hole patterning," *Advanced Materials*, vol. 24, no. 23, pp. 3107–3114, 2012.

[9] Y. Du, D. Guo, M. D. Wong, H. Yi, H.-S. P. Wong, H. Zhang, and Q. Ma, "Block copolymer directed self-assembly (DSA) aware contact layer optimization for 10 nm 1d standard cell library," in *IEEE/ACM International Conference on Computer-Aided Design*, 2013.

[10] D. Q. Pike, F. A. Detcheverry, M. Müller, and J. J. de Pablo, "Theoretically informed coarse grain simulations of polymeric systems," *The Journal of chemical physics*, vol. 131, p. 084 903, 2009.

[11] H. D. Ceniceros and G. H. Fredrickson, "Numerical solution of polymer self-consistent field theory," *Multiscale Modeling & Simulation*, vol. 2, no. 3, pp. 452–474, 2004.

[12] A. Latypov, M. Preil, G. Schmid, J. Xu, H. Yi, K. Yoshimoto, and Y. Zou, "Exploration of the directed self-assembly based nano-fabrication design space using computational simulations," in *SPIE Advanced Lithography*, International Society for Optics and Photonics, 2013, pp. 868 013–868 013.

[13] K. Yoshimoto, B. L. Peters, G. S. Khaira, and J. J. de Pablo, "Scalable simulations for directed self-assembly patterning with the use of GPU parallel computing," in *SPIE Advanced Lithography*, International Society for Optics and Photonics, 2012, 83232P–83232P.

[14] Y. Wei and R. L. Brainard, "Advanced processes for 193-nm immersion lithography," in. SPIE Press Book, 2009, ch. 9, pp. 215–225.

[15] K. Yuan, J.-S. Yang, and D. Pan, "Double patterning layout decomposition for simultaneous conflict and stitch minimization," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 2, pp. 185–196, Feb. 2010, ISSN: 0278-0070.

[16] C.-H. Hsu, Y.-W. Chang, and S. R. Nassif, "Simultaneous layout migration and decomposition for double patterning technology," in *Proceedings of the 2009 International Conference on Computer-Aided Design*, ser. ICCAD '09, San Jose, California: ACM, 2009, pp. 595–600.

[17] Y. Xu and C. Chu, "GREMA: Graph reduction based efficient mask assignment for double patterning technology," in *Proceedings of the 2009 International Conference on Computer-Aided Design*, ser. ICCAD '09, San Jose, California: ACM, 2009, pp. 601–606.

[18] M. Mirsaeedi, J. Torres, and M. Anis, "Self-aligned double-patterning (sadp) friendly detailed routing," in *Proceedings of SPIE*, vol. 7974, 2011, 79740O.

[19] J. Gao and D. Pan, "Flexible self-aligned double patterning aware detailed routing with prescribed layout planning," in *Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design*, ACM, 2012, pp. 25–32.

[20] Y. Ma, J. Sweis, H. Yoshida, Y. Wang, J. Kye, and H. Levinson, "Self-aligned double patterning (SADP) compliant design flow," in *SPIE Advanced Lithography*, International Society for Optics and Photonics, 2012, pp. 832 706–832 706.

[21] Y. Ma, J. Sweis, C. Bencher, H. Dai, Y. Chen, J. P. Cain, Y. Deng, J. Kye, and H. J. Levinson, "Decomposition strategies for self-aligned double patterning," in *Proc. SPIE*, vol. 7641, 76410T, Jan. 2010.

[22] C. Bencher, "SADP: The best option," *Nanochip Technology Journal*, vol. 5, no. 2, pp. 1–6, Oct. 2007.

[23] M. C. Smayling, C. Bencher, H. D. Chen, H. Dai, and M. P. Duane, "APF pitch-halving for 22nm logic cells using gridded design rules," *Proc. SPIE*, vol. 6925, no. 1, 69251E–69251E-8, 2008.

[24] S. Sun, C. Bencher, Y. Chen, H. Dai, M.-P. Cai, J. Jin, P. Blanco, L. Miao, P. Xu, X. Xu, *et al.*, "Demonstration of 32nm half-pitch electrical testable nand flash patterns using self-aligned double patterning," in *SPIE Advanced Lithography*, International Society for Optics and Photonics, 2009, pp. 72740D–72740D-7.

[25] Y.-S. Chang, J.-C. Lai, C.-C. Lin, J. Sweis, and J. Yu, "Full-area pattern decomposition of self-aligned double patterning for 30nm node nand flash process," *Proc. SPIE*, Jan. 2010.

[26] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition for double patterning lithography," in *Proc. ICCAD*, ser. ICCAD '08, San Jose, California: IEEE Press, 2008, pp. 465–472.

[27] Q. Li, "NP-completeness result for positive line-by-fill SADP process," *SPIE Photomask Technology*, vol. 7823, 78233P, 2010.

[28] Z. Xiao, Y. Du, H. Zhang, and M. D. Wong, "A polynomial time exact algorithm for self-aligned double patterning layout decomposition," in *Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design*, ACM, 2012, pp. 17–24.

[29] ——, "A polynomial time exact algorithm for overlay-resistant self-aligned double patterning (SADP) layout decomposition," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 8, pp. 1228–1239, 2013.

[30] H. Zhang, Y. Du, M. Wong, R. Topaloglu, and W. Conley, "Effective decomposition algorithm for self-aligned double patterning lithography," in *Proceedings of SPIE*, vol. 7973, 2011, 79730J.

[31] B. Aspvall, "A linear-time algorithm for testing the truth of certain quantified Boolean formulas," *Inform. Process. Lett.*, vol. 8, pp. 121–123, 1979.

[32] *Nangate open cell library*, http://www.si2.org/openeda.si2.org/projects/nangatelib.

[33] *Gurobi optimizer*, http://www.gurobi.com.

[34] Z. Xiao, Y. Du, H. Tian, and M. D. Wong, "Optimally minimizing overlay violation in self-aligned double patterning decomposition for row-based standard cell layout in polynomial time," in *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, IEEE, 2013, pp. 32–39.

[35] H. Tian, H. Zhang, Q. Ma, Z. Xiao, and M. D. Wong, "A polynomial time triple patterning algorithm for cell based row-structure layout," in *Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on*, IEEE, 2012, pp. 57–64.

[36] H. Zhang, Y. Du, M. D. Wong, and R. O. Topaloglu, "Characterization and decomposition of self-aligned quadruple patterning friendly layout," in *SPIE Advanced Lithography*, International Society for Optics and Photonics, 2012, 83260F–83260F.

[37] X.-Y. Bao, H. Yi, C. Bencher, L.-W. Chang, H. Dai, Y. Chen, P.-T. Chen, and H.-S. Wong, "SRAM, NAND, DRAM contact hole patterning using block copolymer directed self-assembly guided by small topographical templates," in *Electron Devices Meeting (IEDM), 2011 IEEE International*, IEEE, 2011, pp. 7–7.

[38] C. Bencher, H. Dai, and Y. Chen, "Gridded design rule scaling: Taking the CPU toward the 16nm node," in *Proc. SPIE*, vol. 7274, 2009, 72740G.

[39] Z. Xiao, Y. Du, M. D. F. Wong, and H. Zhang, "DSA template mask determination and cut redistribution for advanced 1D gridded design," in *SPIE Photomask Technology*, International Society for Optics and Photonics, 2013, p. 888 017.

[40] Z. Xiao, Y. Du, H. Tian, and M. D. F. Wong, "Dsa template optimization for contact layer in 1d standard cell design," in *SPIE Advanced Lithography*, vol. 9049, 2014.

[41] H. Yi and H.-S. P. Wong, "Block copolymer directed self-assembly two-hole pattern inside peanut-shaped templates," in *EIPBN*, 2013, 10B–05.

[42] S. Kirkpatrick, D. G. Jr., and M. P. Vecchi, "Optimization by simmulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[43] R. A. Rutenbar, "Simulated annealing algorithms: An overview," *Circuits and Devices Magazine, IEEE*, vol. 5, no. 1, pp. 19–26, 1989.

[44] R. E. Tarjan, "Efficiency of a good but not linear set union algorithm," *Journal of the ACM (JACM)*, vol. 22, no. 2, pp. 215–225, 1975.

[45] D. Hill, *Method and system for high speed detailed placement of cells within an integrated circuit design*, US Patent 6,370,673, Apr. 2002.

[46] Y. Borodovsky, "Marching to the beat of Moore's law," in *SPIE 31st International Symposium on Advanced Lithography*, International Society for Optics and Photonics, 2006, pp. 615 301–615 301.

[47] Y. Ma, J. A. Torres, G. Fenger, Y. Granik, J. Ryckaert, G. Vanderberghe, J. Bekaert, and J. Word, "Challenges and opportunities in applying grapho-epitaxy DSA lithography to metal cut and contact/via applications," in *30th European Mask and Lithography Conference*, International Society for Optics and Photonics, 2014, 92310T–92310T.

[48] H. Tian, Y. Du, H. Zhang, Z. Xiao, and M. Wong, "Triple patterning aware detailed placement with constrained pattern assignment," in *Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on*, 2014.

[49] Y. Du, H. Zhang, M. D. Wong, and K.-Y. Chao, "Hybrid lithography optimization with e-beam and immersion processes for 16nm 1d gridded design," in *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, IEEE, 2012, pp. 707–712.

[50] J.-R. Gao, B. Yu, and D. Z. Pan, "Self-aligned double patterning layout decomposition with complementary e-beam lithography.," in *ASP-DAC*, 2014, pp. 143–148.

[51] H. Tian, H. Zhang, Z. Xiao, and M. D. Wong, "Hybrid lithography for triple patterning decomposition and e-beam lithography," in *SPIE Advanced Lithography*, International Society for Optics and Photonics, 2014, 90520P–90520P.

[52] Y. Du, Z. Xiao, M. D. F. Wong, H. Yi, and H.-S. P. Wong, "DSA-aware detailed routing for via layer optimization," vol. 9049, 2014, 90492J–90492J-8.

[53] J. A. Torres, K. Sakajiri, D. Fryer, Y. Granik, Y. Ma, P. Krasnova, G. Fenger, S. Nagahara, S. Kawakami, B. Rathsack, *et al.*, "Physical verification and manufacturing of contact/via layers using grapho-epitaxy dsa processes," in *SPIE Advanced Lithography*, International Society for Optics and Photonics, 2014, 90530R–90530R.

[54] Z. Xiao, Y. Du, H. Tian, M. D. F. Wong, H. Yi, H.-S. P. Wong, and H. Zhang, "Directed self-assembly (dsa) template pattern verification," in *DAC '14: Proceedings of the 51th Annual Design Automation Conference*, Jun. 2014.

[55] Z. Xiao, Y. Du, M. D. Wong, H. Yi, H. Wong, and H. Zhang, "Contact pitch and location prediction for directed self-assembly template verification," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, IEEE, 2015, pp. 644–651.

[56] Y. Badr, J. Torres, Y. Ma, J. Mitra, and P. Gupta, "Incorporating dsa in multipatterning semiconductor manufacturing technologies," in *SPIE Advanced Lithography*, International Society for Optics and Photonics, 2015, 94270P–94270P.

[57] Z. Xiao, C.-X. Lin, H. Zhang, and M. D. Wong, "Contact layer decomposition to enable dsa with multi-patterning technique for standard cell based layout," in *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*, IEEE, 2016.

[58] P. Heggernes, P. V. Hof, D. Lokshtanov, and C. Paul, "Obtaining a bipartite graph by contracting few edges," *SIAM Journal on Discrete Mathematics*, vol. 27, no. 4, pp. 2143–2156, 2013.

[59] M. Samer and S. Szeider, "Fixed-parameter tractability," *Handbook of Satisfiability*, 2009.

[60] S. Nangate, *California (2008). 45 nm open cell library*, http://www.nangate.com, 2008.

[61] H. Yi, A. Latypov, and H.-S. P. Wong, "Computational simulation of block copolymer directed self-assembly in small topographical guiding templates," in *SPIE Advanced Lithography*, International Society for Optics and Photonics, 2013, pp. 86801L–86801L.

[62] N. Ma, J. Ghan, S. Mishra, C. Spanos, K. Poolla, N. Rodriguez, and L. Capodieci, "Automatic hotspot classification using pattern-based clustering," in *Advanced Lithography*, International Society for Optics and Photonics, 2008, pp. 692 505–692 505.

[63] D. G. Drmanac, F. Liu, and L.-C. Wang, "Predicting variability in nanoscale lithography processes," in *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*, IEEE, 2009, pp. 545–550.

[64] J.-Y. Wuu, F. G. Pikus, A. Torres, and M. Marek-Sadowska, "Detecting context sensitive hotspots in standard cell libraries," in *Proc. SPIE*, vol. 7275, 2009, p. 727 515.

[65] Y.-T. Yu, G.-H. Lin, I. H.-R. Jiang, and C. Chiang, "Machine-learning-based hotspot detection using topological classification and critical feature extraction," in *DAC '13: Proceedings of the 50th Annual Design Automation Conference*, May 2013.

[66] S.-Y. Lin, J.-Y. Chen, J.-C. Li, W.-y. Wen, and S.-C. Chang, "A novel fuzzy matching model for lithography hotspot detection," in *DAC '13: Proceedings of the 50th Annual Design Automation Conference*, May 2013.

[67] D. Ding, A. J. Torres, F. G. Pikus, and D. Z. Pan, "High performance lithographic hotspot detection using hierarchically refined machine learning," in *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, IEEE Press, 2011, pp. 775–780.

[68] Z. Xiao, D. Guo, M. D. Wong, H. Yi, M. C. Tung, and H.-S. P. Wong, "Layout optimization and template pattern verification for directed self-assembly (DSA)," in *Proceedings of the 52nd Annual Design Automation Conference*, ACM, 2015, p. 199.

[69] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 26, no. 1, pp. 43–49, 1978.

[70] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, IEEE, vol. 1, 2005, pp. 886–893.

[71] M. T. Hagan, H. B. Demuth, M. H. Beale, *et al.*, *Neural network design*. Pws Pub. Boston, 1996.

[72] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[73] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *Computational learning theory*, Springer, 1995, pp. 23–37.

[74] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, pp. 121–167, 1998.

[75] D. H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[76] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, 27:1–27:27, 3 2011, Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.